

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	R19-11

Document Change History			
Date	Release	Changed by	Description
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none">• Introduced<ul style="list-style-type: none">– 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Introduced Adaptive Core types• Introduced exception-less API• Refined DDS network binding• Minor changes and bugfixes

2018-03-29	18-03	AUTOSAR Release Management	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1	Introduction and functional overview	8
2	Acronyms and Abbreviations	9
3	Related documentation	10
3.1	Input documents & related standards and norms	10
3.2	Related specification	11
4	Constraints and assumptions	12
4.1	Limitations	12
4.2	Applicability to car domains	12
5	Dependencies to other functional clusters	13
5.1	Platform dependencies	13
6	Requirements Tracing	14
7	Functional specification	43
7.1	General description	43
7.1.1	Architectural concepts	43
7.1.2	Design decisions	45
7.1.3	Communication paradigms	46
7.1.4	Service contract versioning	47
7.2	End-to-end communication protection for Events	49
7.2.1	Limitations	49
7.2.2	Publisher	50
7.2.3	Subscriber - GetNewSamples	52
7.2.3.1	Case 1 - there are one or more serialized samples	54
7.2.3.2	Case 2 - there are no serialized samples	54
7.2.4	Subscriber - Callback f	55
7.2.5	Subscriber - Access to E2E information	55
7.3	End-to-end communication protection for Methods	55
7.3.1	Limitations	55
7.3.2	Call a service method (Client)	56
7.3.3	Processing of service methods (Server)	59
7.3.4	E2E error handler	61
7.4	Raw Data Streaming	62
7.4.1	Raw Data Streaming Interface	62
7.4.1.1	Limitations	63
7.4.2	Raw Data Streaming	63
7.5	Network binding	64
7.5.1	SOME/IP Network binding	66
7.5.1.1	Service Discovery	67
7.5.1.2	Accumulation of SOME/IP messages	74
7.5.1.3	Execution context of message reception actions	76

7.5.1.4	Handling Events	76
7.5.1.5	Handling Method Calls	79
7.5.1.6	Handling Fields	88
7.5.1.7	Serialization of Payload	96
7.5.1.8	Marker Interface	117
7.5.2	Signal-Based Network binding	118
7.5.2.1	Service Discovery	122
7.5.2.2	Accumulation of messages	130
7.5.2.3	Execution context of message reception actions	131
7.5.2.4	Handling Events	131
7.5.2.5	Handling Method Calls	136
7.5.2.6	Handling Fields	145
7.5.2.7	Serialization of Payload	154
7.5.3	DDS Network binding	155
7.5.3.1	Service Discovery	155
7.5.3.2	Handling Events	165
7.5.3.3	Handling Method Calls	170
7.5.3.4	Handling Fields	180
7.5.3.5	Serialization of Payload	192
7.6	Security	194
7.6.1	Access Control	194
7.6.2	Secure Communication	196
7.6.2.1	SOME/IP Network binding	196
7.6.2.2	DDS	203
7.6.2.3	Raw Data Streaming	205
7.7	Communication API	206
7.7.1	Offer service	206
7.7.2	Service skeleton creation	207
7.7.3	Processing of service methods	207
7.7.4	Registering get handlers for fields	208
7.7.5	Registering set handlers for fields	208
7.7.6	Find service	209
7.7.7	Receive events	209
7.7.7.1	Receive event by polling	209
7.7.7.2	Receive event by getting triggered	209
7.7.8	Call a service method	210
7.7.9	Update notification events for fields	210
7.7.10	Instance Specifier Translation	211
7.7.11	Invalid Value	211
8	Communication API specification	212
8.1	C++ language binding	212
8.1.1	API Header files	212
8.1.1.1	Service header files	212
8.1.1.2	Common header file	215
8.1.1.3	Types header file	216

8.1.1.4	Implementation Types header files	217
8.1.1.5	Raw data stream header file	218
8.1.2	API Data Types	219
8.1.2.1	Service Identifier Data Types	219
8.1.2.2	Event Related Data Types	223
8.1.2.3	Method Related Data Types	226
8.1.2.4	Generic Data Types	226
8.1.2.5	Communication Payload Data Types	237
8.1.2.6	Error Types	260
8.1.2.7	E2E Related Data Types	262
8.1.3	API Reference	265
8.1.3.1	Object Creation via Construction Token	266
8.1.3.2	Offer service	267
8.1.3.3	Service skeleton creation	267
8.1.3.4	Send event	271
8.1.3.5	Provide a service method	272
8.1.3.6	Processing of service methods	273
8.1.3.7	Registering get handlers for fields	274
8.1.3.8	Registering set handlers for fields	274
8.1.3.9	Find service	275
8.1.3.10	Service proxy creation	277
8.1.3.11	Service proxy destruction	278
8.1.3.12	Service event subscription	278
8.1.3.13	Receive event	280
8.1.3.14	Receive event by getting triggered	282
8.1.3.15	Call a service method	283
8.1.3.16	Get method for fields	286
8.1.3.17	Set method for fields	287
8.1.3.18	Instance Specifier Translation	287
8.1.3.19	Raw Data Stream API	287
A	Mentioned Class Tables	291
B	History of Specification Items	360
B.1	Constraint and Specification Item History of this document according to AUTOSAR Release 17-10	360
B.1.1	Added Traceables in 17-10	360
B.1.2	Changed Traceables in 17-10	365
B.1.3	Deleted Traceables in 17-10	366
B.2	Constraint and Specification Item History of this document according to AUTOSAR Release 18-03	367
B.2.1	Added Traceables in 18-03	367
B.2.2	Changed Traceables in 18-03	370
B.2.3	Deleted Traceables in 18-03	377
B.3	Constraint and Specification Item History of this document according to AUTOSAR Release 18-10	377
B.3.1	Added Traceables in 18-10	377

B.3.2	Changed Traceables in 18-10	382
B.3.3	Deleted Traceables in 18-10	387
B.4	Constraint and Specification Item History of this document according to AUTOSAR Release 19-03	389
B.4.1	Added Traceables in 19-03	389
B.4.2	Changed Traceables in 19-03	389
B.4.3	Deleted Traceables in 19-03	389
B.5	Constraint and Specification Item History of this document according to AUTOSAR Release 19-11	390
B.5.1	Added Traceables in R19-11	390
B.5.2	Changed Traceables in R19-11	395
B.5.3	Deleted Traceables in R19-11	402

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 build-in safety mechanism (E2E protection), which can be used for all levels of communication for events that are received using polling.

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 <code>INVOK</code> operation (used by, e.g., <code>std::function</code> , <code>std::bind</code> , and <code>std::thread::thread</code>) is applicable. This operation may be performed explicitly using the library function <code>std::invoke</code> . (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.: <ul style="list-style-type: none">• 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] SOME/IP Protocol Specification
AUTOSAR_PRS_SOMEIPProtocol
- [5] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification
- [6] Requirements on E2E
AUTOSAR_RS_E2E
- [7] E2E Protocol Specification
AUTOSAR_PRS_E2EProtocol
- [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 Core Types for Adaptive Platform
AUTOSAR_SWS_CoreTypes
- [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] DDS Security, Version 1.1
<https://www.omg.org/spec/DDS-SECURITY/1.1>
- [25] Methodology for Adaptive Platform
AUTOSAR_TR_AdaptiveMethodology
- [26] General Specification of Adaptive Platform
AUTOSAR_SWS_General
- [27] ISO/IEC 14882:2011, Information technology – Programming languages – C++
<http://www.iso.org>
- [28] N4659: Working Draft, Standard for ProgrammingLanguage C++
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf>
- [29] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

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 limitations regarding E2E protection and the detectable failure modes are described in chapter [7.3.1](#).

The following limitations regarding optionality introduced with the Tag-Length-Value serialization principle described in [4] and [5] apply:

- **Optional method arguments**

The Specification does not support the existence of optional method arguments.

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 [6] and [7]. The E2E interfaces 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_00113]	No description	[SWS_CM_00002] [SWS_CM_00003] [SWS_CM_00004] [SWS_CM_00005] [SWS_CM_00006] [SWS_CM_00007] [SWS_CM_00008] [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_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_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_00383] [SWS_CM_00402] [SWS_CM_00403] [SWS_CM_00404] [SWS_CM_00405] [SWS_CM_00406] [SWS_CM_00407] [SWS_CM_00409] [SWS_CM_00410]

Requirement	Description	Satisfied by
		[SWS_CM_00414] [SWS_CM_00424] [SWS_CM_00425] [SWS_CM_00449] [SWS_CM_00502] [SWS_CM_00503] [SWS_CM_00504] [SWS_CM_00505] [SWS_CM_00506] [SWS_CM_00507] [SWS_CM_00508] [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_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_01032] [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_10375] [SWS_CM_10383] [SWS_CM_10392] [SWS_CM_10393] [SWS_CM_10394] [SWS_CM_10395] [SWS_CM_10396] [SWS_CM_10397] [SWS_CM_10398] [SWS_CM_10399] [SWS_CM_10400] [SWS_CM_10401] [SWS_CM_10402] [SWS_CM_10403] [SWS_CM_10404] [SWS_CM_10405] [SWS_CM_10406] [SWS_CM_10407] [SWS_CM_10408] [SWS_CM_10409] [SWS_CM_10433] [SWS_CM_10434] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11266] [SWS_CM_90420] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90424] [SWS_CM_90434] [SWS_CM_90435] [SWS_CM_90437] [SWS_CM_90438]

Requirement	Description	Satisfied by
[RS_AP_00114]	C++ interface shall be compatible with C++11.	[SWS_CM_00002] [SWS_CM_00003] [SWS_CM_00004] [SWS_CM_00005] [SWS_CM_00006] [SWS_CM_00007] [SWS_CM_00008] [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_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_00383] [SWS_CM_00402] [SWS_CM_00403] [SWS_CM_00404] [SWS_CM_00405] [SWS_CM_00406] [SWS_CM_00407] [SWS_CM_00409] [SWS_CM_00410] [SWS_CM_00414] [SWS_CM_00424] [SWS_CM_00425] [SWS_CM_00449] [SWS_CM_00502] [SWS_CM_00503] [SWS_CM_00504] [SWS_CM_00505] [SWS_CM_00506] [SWS_CM_00507] [SWS_CM_00508] [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_01001] [SWS_CM_01002] [SWS_CM_01004]

Requirement	Description	Satisfied by
		[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_01032] [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_10375] [SWS_CM_10383] [SWS_CM_10392] [SWS_CM_10393] [SWS_CM_10394] [SWS_CM_10395] [SWS_CM_10396] [SWS_CM_10397] [SWS_CM_10398] [SWS_CM_10399] [SWS_CM_10400] [SWS_CM_10401] [SWS_CM_10402] [SWS_CM_10403] [SWS_CM_10404] [SWS_CM_10405] [SWS_CM_10406] [SWS_CM_10407] [SWS_CM_10408] [SWS_CM_10409] [SWS_CM_10433] [SWS_CM_10434] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11266] [SWS_CM_90420] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90424] [SWS_CM_90434] [SWS_CM_90435] [SWS_CM_90437] [SWS_CM_90438]
[RS_AP_00115]	Namespaces.	[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_90421] [SWS_CM_90422] [SWS_CM_90424] [SWS_CM_90438]
[RS_AP_00116]	Header file name.	[SWS_CM_01002] [SWS_CM_01012] [SWS_CM_01013] [SWS_CM_10373]

Requirement	Description	Satisfied by
[RS_AP_00119]	Return values / application errors.	[SWS_CM_00118] [SWS_CM_00122] [SWS_CM_00123] [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_90421] [SWS_CM_90422]
[RS_AP_00120]	Method and Function names.	[SWS_CM_00101] [SWS_CM_00111] [SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00118] [SWS_CM_00119] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00125] [SWS_CM_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_00309] [SWS_CM_00311] [SWS_CM_00316] [SWS_CM_00333] [SWS_CM_00334] [SWS_CM_00383] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_90420] [SWS_CM_90435] [SWS_CM_90437] [SWS_CM_90438]
[RS_AP_00121]	Parameter names.	[SWS_CM_00113] [SWS_CM_00118] [SWS_CM_00119] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00125] [SWS_CM_00130] [SWS_CM_00131] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00162] [SWS_CM_00181] [SWS_CM_00333] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [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_00504] [SWS_CM_01050] [SWS_CM_10392]
[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_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_00302] [SWS_CM_00403] [SWS_CM_00406] [SWS_CM_00407] [SWS_CM_00409] [SWS_CM_00449] [SWS_CM_00503] [SWS_CM_00505] [SWS_CM_00509] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_00704] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_01032] [SWS_CM_01050] [SWS_CM_10362] [SWS_CM_10434] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11266]
[RS_AP_00128]	Error reporting.	[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_00701] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_10434] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438]
[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_12367]
[RS_AP_00132]	noexcept behavior of API functions	[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_10434] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_90420]
[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_00118] [SWS_CM_00152] [SWS_CM_00350] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_10436] [SWS_CM_10450] [SWS_CM_10452] [SWS_CM_10590]

Requirement	Description	Satisfied by
[RS_AP_00138]	Return type of asynchronous function calls.	[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] [SWS_CM_00196] [SWS_CM_00197] [SWS_CM_00199] [SWS_CM_00199] [SWS_CM_10414]
[RS_AP_00139]	Return type of synchronous function calls.	[SWS_CM_00195] [SWS_CM_00701] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_10434] [SWS_CM_10434] [SWS_CM_10435] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10438]
[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_10373] [SWS_CM_10374] [SWS_CM_10453] [SWS_CM_10488] [SWS_CM_10490]
[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_10375] [SWS_CM_10489]
[RS_CM_00003]	The Communication Management shall define how language specific data types are derived from modeled data types.	[SWS_CM_00400] [SWS_CM_00402] [SWS_CM_00403] [SWS_CM_00404] [SWS_CM_00405] [SWS_CM_00406] [SWS_CM_00407] [SWS_CM_00408] [SWS_CM_00409] [SWS_CM_00410] [SWS_CM_00411] [SWS_CM_00414] [SWS_CM_00421] [SWS_CM_00423] [SWS_CM_00424] [SWS_CM_00425] [SWS_CM_00426] [SWS_CM_00450] [SWS_CM_00452] [SWS_CM_00503] [SWS_CM_00504] [SWS_CM_00505] [SWS_CM_00509] [SWS_CM_01032] [SWS_CM_10376] [SWS_CM_10392] [SWS_CM_10393] [SWS_CM_10394] [SWS_CM_10395] [SWS_CM_10396] [SWS_CM_10397] [SWS_CM_10398] [SWS_CM_10399] [SWS_CM_10400] [SWS_CM_10401] [SWS_CM_10402] [SWS_CM_10403] [SWS_CM_10404] [SWS_CM_10405] [SWS_CM_10406] [SWS_CM_10407] [SWS_CM_10408] [SWS_CM_10409]

Requirement	Description	Satisfied by
[RS_CM_00004]	Communication Management shall support the translation between signal-based and service-oriented communication	[SWS_CM_80001] [SWS_CM_80002] [SWS_CM_80003] [SWS_CM_80004] [SWS_CM_80005] [SWS_CM_80006] [SWS_CM_80007] [SWS_CM_80008] [SWS_CM_80009] [SWS_CM_80010] [SWS_CM_80012] [SWS_CM_80013] [SWS_CM_80014] [SWS_CM_80015] [SWS_CM_80016] [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_80029] [SWS_CM_80030] [SWS_CM_80031] [SWS_CM_80032] [SWS_CM_80033] [SWS_CM_80034] [SWS_CM_80035] [SWS_CM_80036] [SWS_CM_80037] [SWS_CM_80038] [SWS_CM_80039] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80043] [SWS_CM_80044] [SWS_CM_80045] [SWS_CM_80046] [SWS_CM_80047] [SWS_CM_80048] [SWS_CM_80049] [SWS_CM_80050] [SWS_CM_80051] [SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80054] [SWS_CM_80055] [SWS_CM_80056] [SWS_CM_80057] [SWS_CM_80058] [SWS_CM_80059] [SWS_CM_80060] [SWS_CM_80061] [SWS_CM_80062] [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_80071] [SWS_CM_80072] [SWS_CM_80073] [SWS_CM_80074] [SWS_CM_80075] [SWS_CM_80076] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80081] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80085] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088] [SWS_CM_80089] [SWS_CM_80090] [SWS_CM_80091] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094] [SWS_CM_80095] [SWS_CM_80096] [SWS_CM_80097] [SWS_CM_80098] [SWS_CM_80099] [SWS_CM_80100] [SWS_CM_80101] [SWS_CM_80102] [SWS_CM_80103]

Requirement	Description	Satisfied by
[RS_CM_00101]	Communication Management shall provide an interface to offer services	[SWS_CM_00002] [SWS_CM_00101] [SWS_CM_00102] [SWS_CM_00103] [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_00350] [SWS_CM_09004] [SWS_CM_10410] [SWS_CM_10433] [SWS_CM_10434] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10450] [SWS_CM_10451] [SWS_CM_10458] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11004] [SWS_CM_11029] [SWS_CM_11030] [SWS_CM_11031] [SWS_CM_80005] [SWS_CM_80008]
[RS_CM_00102]	Communication Management shall provide an interface to find services	[SWS_CM_00004] [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_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_80006] [SWS_CM_80007]
[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_00141] [SWS_CM_00205] [SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_00700] [SWS_CM_10377] [SWS_CM_10381] [SWS_CM_11018] [SWS_CM_11019] [SWS_CM_11020] [SWS_CM_11133] [SWS_CM_11134] [SWS_CM_11135] [SWS_CM_80010] [SWS_CM_80011] [SWS_CM_80012]
[RS_CM_00104]	Communication Management shall provide an interface to stop the subscription to an event of a service instance	[SWS_CM_00151] [SWS_CM_00207] [SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_10378] [SWS_CM_11021] [SWS_CM_11136] [SWS_CM_80015] [SWS_CM_80016]
[RS_CM_00105]	Communication Management shall provide an interface to stop offering services	[SWS_CM_00111] [SWS_CM_00204] [SWS_CM_11005] [SWS_CM_80009]

Requirement	Description	Satisfied by
[RS_CM_00106]	Communication Management shall provide a means to monitor the state of the subscription to an event	[SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_00316] [SWS_CM_00333] [SWS_CM_00334] [SWS_CM_11022] [SWS_CM_11027] [SWS_CM_11028] [SWS_CM_11137] [SWS_CM_11142] [SWS_CM_11143]
[RS_CM_00107]	Communication Management shall provide a means to automatically update a proxy instance in case of restart of the offered service	[SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_10383]
[RS_CM_00200]	The Communication Management shall transform Fully Qualified Service IDs to communication protocol specific Service IDs	[SWS_CM_00102] [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_10463] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11004] [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_11151] [SWS_CM_80007] [SWS_CM_80008] [SWS_CM_80010] [SWS_CM_80011] [SWS_CM_80012] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80029] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80071] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80094] [SWS_CM_90403]

Requirement	Description	Satisfied by
[RS_CM_00201]	Communication Management shall provide an API to send events to other applications	[SWS_CM_00003] [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_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_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] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10326] [SWS_CM_10361]

Requirement	Description	Satisfied by
		[SWS_CM_10391] [SWS_CM_10459] [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_80029] [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_80071] [SWS_CM_80074] [SWS_CM_90437] [SWS_CM_90438]
[RS_CM_002013]	No description	[SWS_CM_11108]
[RS_CM_00202]	Communication Management shall provide an API to the application to poll received events	[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] [SWS_CM_10227] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10242]

Requirement	Description	Satisfied by
		[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_11023] [SWS_CM_11024] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11138] [SWS_CM_11139] [SWS_CM_11262] [SWS_CM_11263] [SWS_CM_80034] [SWS_CM_80076]
[RS_CM_00203]	Communication Management shall trigger the application on reception of an event	[SWS_CM_00181] [SWS_CM_00182] [SWS_CM_00183] [SWS_CM_00306] [SWS_CM_00309] [SWS_CM_00709] [SWS_CM_00710] [SWS_CM_00711] [SWS_CM_10296] [SWS_CM_10328] [SWS_CM_10379] [SWS_CM_10380] [SWS_CM_10470] [SWS_CM_11025] [SWS_CM_11026] [SWS_CM_11140] [SWS_CM_11141] [SWS_CM_80030] [SWS_CM_80031] [SWS_CM_80072] [SWS_CM_80073]
[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]

Requirement	Description	Satisfied by
		[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_10234] [SWS_CM_10235] [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] [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]

Requirement	Description	Satisfied by
		[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_10361] [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_10459] [SWS_CM_11000] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11004] [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]

Requirement	Description	Satisfied by
		[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_80001] [SWS_CM_80002] [SWS_CM_80003] [SWS_CM_80005] [SWS_CM_80006] [SWS_CM_80007] [SWS_CM_80008] [SWS_CM_80009] [SWS_CM_80010] [SWS_CM_80011] [SWS_CM_80012] [SWS_CM_80013] [SWS_CM_80014] [SWS_CM_80015] [SWS_CM_80016] [SWS_CM_80017] [SWS_CM_80018] [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_80029] [SWS_CM_80030] [SWS_CM_80031] [SWS_CM_80032] [SWS_CM_80034] [SWS_CM_80035] [SWS_CM_80036] [SWS_CM_80037] [SWS_CM_80038] [SWS_CM_80039] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80043] [SWS_CM_80044] [SWS_CM_80045] [SWS_CM_80046] [SWS_CM_80047] [SWS_CM_80048] [SWS_CM_80049] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80054] [SWS_CM_80055] [SWS_CM_80056] [SWS_CM_80057] [SWS_CM_80058] [SWS_CM_80059] [SWS_CM_80060] [SWS_CM_80061] [SWS_CM_80062] [SWS_CM_80063] [SWS_CM_80064] [SWS_CM_80065]

Requirement	Description	Satisfied by
		[SWS_CM_80066] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80071] [SWS_CM_80072] [SWS_CM_80073] [SWS_CM_80074] [SWS_CM_80076] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80081] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80085] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088] [SWS_CM_80089] [SWS_CM_80090] [SWS_CM_80091] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094] [SWS_CM_80095] [SWS_CM_80096] [SWS_CM_80097] [SWS_CM_80098] [SWS_CM_80099] [SWS_CM_90443] [SWS_CM_90444] [SWS_CM_90445] [SWS_CM_90446] [SWS_CM_90451] [SWS_CM_90452]
[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_01032] [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_00207]	No description	[SWS_CM_00118] [SWS_CM_10452] [SWS_CM_10590]
[RS_CM_00211]	Communication Management shall provide an interface to provide methods to other applications	[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_00400] [SWS_CM_00402] [SWS_CM_00403] [SWS_CM_00404] [SWS_CM_00405] [SWS_CM_00406] [SWS_CM_00407] [SWS_CM_00408] [SWS_CM_00409]

Requirement	Description	Satisfied by
		[SWS_CM_00410] [SWS_CM_00411] [SWS_CM_00414] [SWS_CM_00421] [SWS_CM_00423] [SWS_CM_00424] [SWS_CM_00425] [SWS_CM_00426] [SWS_CM_00449] [SWS_CM_00450] [SWS_CM_00452] [SWS_CM_00503] [SWS_CM_00504] [SWS_CM_00505] [SWS_CM_00509] [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_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_10376] [SWS_CM_10391] [SWS_CM_10409] [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]

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_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_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_80035] [SWS_CM_80036] [SWS_CM_80037] [SWS_CM_80038] [SWS_CM_80039] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80043] [SWS_CM_80044] [SWS_CM_80045] [SWS_CM_80046] [SWS_CM_80047] [SWS_CM_80048] [SWS_CM_80049] [SWS_CM_80050]

Requirement	Description	Satisfied by
		[SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80054] [SWS_CM_80056] [SWS_CM_80057] [SWS_CM_80061] [SWS_CM_80062] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80081] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80085] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088] [SWS_CM_80089] [SWS_CM_80090] [SWS_CM_80091] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094] [SWS_CM_80095] [SWS_CM_80096] [SWS_CM_80097] [SWS_CM_80098] [SWS_CM_80099]
[RS_CM_00213]	Communication Management shall provide an interface to call service methods asynchronously	[SWS_CM_00006] [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_11100] [SWS_CM_11101] [SWS_CM_11102] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11107] [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]

Requirement	Description	Satisfied by
		[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_80035] [SWS_CM_80036] [SWS_CM_80037] [SWS_CM_80038] [SWS_CM_80039] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80043] [SWS_CM_80044] [SWS_CM_80045] [SWS_CM_80046] [SWS_CM_80047] [SWS_CM_80048] [SWS_CM_80049] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80054] [SWS_CM_80056] [SWS_CM_80057] [SWS_CM_80061] [SWS_CM_80062] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80081] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80085] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088] [SWS_CM_80089] [SWS_CM_80090] [SWS_CM_80091] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094] [SWS_CM_80095] [SWS_CM_80096] [SWS_CM_80097] [SWS_CM_80098] [SWS_CM_80099]
[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]
[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] [SWS_CM_80061] [SWS_CM_80062] [SWS_CM_80098] [SWS_CM_80099]
[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]
[RS_CM_00217]	Communication Management shall provide a method to remotely set the field value	[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] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80082] [SWS_CM_80084] [SWS_CM_80092] [SWS_CM_80094]

Requirement	Description	Satisfied by
[RS_CM_00218]	Communication Management shall provide a method to remotely get the field value	[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] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80082] [SWS_CM_80084] [SWS_CM_80092] [SWS_CM_80094]
[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] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088]
[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] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088]
[RS_CM_00222]	No description	[SWS_CM_10460] [SWS_CM_10461] [SWS_CM_10462] [SWS_CM_10463] [SWS_CM_10464] [SWS_CM_10465] [SWS_CM_10466] [SWS_CM_10468] [SWS_CM_10469] [SWS_CM_90417]
[RS_CM_00223]	Communication Management shall protect the transmission of data using E2E protocol, hidden behind the event API.	[SWS_CM_10461] [SWS_CM_90433]
[RS_CM_00225]	Communication Management shall provide an interface to call fire&forget service methods	[SWS_CM_10475] [SWS_CM_90431] [SWS_CM_90434] [SWS_CM_90435] [SWS_CM_90436]
[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_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]

Requirement	Description	Satisfied by
[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]
[RS_CM_00412]	The Communication Management shall provide TCP/IP Sockets as network protocol for Raw Data Streams	[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]
[RS_CM_00500]	Service Contract Version for a Service Interface	[SWS_CM_01010] [SWS_CM_09004] [SWS_CM_99003]
[RS_CM_00501]	Service Contract Versioning for all Transport Deployment Protocols	[SWS_CM_09004] [SWS_CM_99003]
[RS_CM_00700]	The Service Discovery shall evaluate the service version compatibility for service connection	[SWS_CM_99003]
[RS_CM_00701]	Service Versioning Blacklist	[SWS_CM_10202]
[RS_CM_200]	No description	[SWS_CM_11112]
[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_90431]
[RS_E2E_08540]	E2E protocol shall support protected periodic/mixed periodic communication	[SWS_CM_10460] [SWS_CM_10461] [SWS_CM_10462] [SWS_CM_10463] [SWS_CM_10464] [SWS_CM_10465] [SWS_CM_10466] [SWS_CM_10468] [SWS_CM_10469] [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_90430] [SWS_CM_90433]
[RS_SEC_03002]	No description	[SWS_CM_90001] [SWS_CM_90002] [SWS_CM_90003] [SWS_CM_90005] [SWS_CM_90006] [SWS_CM_90007]
[RS_SEC_03003]	No description	[SWS_CM_90004]
[RS_SEC_03005]	No description	[SWS_CM_90004]
[RS_SEC_03008]	No description	[SWS_CM_90001] [SWS_CM_90002] [SWS_CM_90003] [SWS_CM_90005] [SWS_CM_90006] [SWS_CM_90007]
[RS_SEC_03010]	No description	[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_90101] [SWS_CM_90102] [SWS_CM_90103] [SWS_CM_90104] [SWS_CM_90105] [SWS_CM_90106] [SWS_CM_90107] [SWS_CM_90108] [SWS_CM_90109] [SWS_CM_90110] [SWS_CM_90115] [SWS_CM_90116] [SWS_CM_90117] [SWS_CM_90118] [SWS_CM_90120] [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_90210] [SWS_CM_90211] [SWS_CM_90212] [SWS_CM_90213] [SWS_CM_90214] [SWS_CM_90215]
[RS_SEC_04003]	The assignment of communication to specific secure channels shall be configurable	[SWS_CM_90102] [SWS_CM_90202] [SWS_CM_90212]
[RS_SEC_04004]	Using secure channels shall be transparent on the communication API	[SWS_CM_90111] [SWS_CM_90112] [SWS_CM_90113] [SWS_CM_90114] [SWS_CM_90119]
[RS_SEC_05019]	Access to Adaptive AUTOSAR Foundation and Services	[SWS_CM_90004]
[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] [SWS_CM_80007] [SWS_CM_80008] [SWS_CM_80009] [SWS_CM_80010] [SWS_CM_80011] [SWS_CM_80012] [SWS_CM_80013] [SWS_CM_80014] [SWS_CM_80015] [SWS_CM_80016]
[RS_SOMEIPSD_-00015]	SOME/IP Service Discovery Protocol shall support to subscribe to events	[SWS_CM_00206] [SWS_CM_80013]
[RS_SOMEIPSD_-00016]	SOME/IP Service Discovery Protocol shall support to deny subscriptions	[SWS_CM_00208] [SWS_CM_80014]
[RS_SOMEIPSD_-00024]	SOME/IP Service Discovery shall support configurable timings	[SWS_CM_00201] [SWS_CM_00209] [SWS_CM_80005] [SWS_CM_80006]
[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_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80082]

Requirement	Description	Satisfied by
		[SWS_CM_80083] [SWS_CM_80092] [SWS_CM_80093]
[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_10379] [SWS_CM_10380] [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_80029] [SWS_CM_80030] [SWS_CM_80031] [SWS_CM_80032] [SWS_CM_80034] [SWS_CM_80063] [SWS_CM_80064] [SWS_CM_80065] [SWS_CM_80066] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80071] [SWS_CM_80072] [SWS_CM_80073] [SWS_CM_80074] [SWS_CM_80076]
[RS_SOMEIP_00005]	SOME/IP protocol shall support different strategies for event communication	[SWS_CM_10034] [SWS_CM_10287] [SWS_CM_10319] [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] [SWS_CM_80035] [SWS_CM_80036] [SWS_CM_80037] [SWS_CM_80039] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80043] [SWS_CM_80044] [SWS_CM_80045] [SWS_CM_80053]
[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_80035] [SWS_CM_80036] [SWS_CM_80037] [SWS_CM_80039] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80043] [SWS_CM_80044] [SWS_CM_80045] [SWS_CM_80046] [SWS_CM_80047] [SWS_CM_80048] [SWS_CM_80049] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80056] [SWS_CM_80057] [SWS_CM_80061] [SWS_CM_80062] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80081] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80085] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088] [SWS_CM_80089] [SWS_CM_80090] [SWS_CM_80091] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094] [SWS_CM_80096] [SWS_CM_80097] [SWS_CM_80098] [SWS_CM_80099]
[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_80027] [SWS_CM_80028] [SWS_CM_80041] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80055] [SWS_CM_80058] [SWS_CM_80059] [SWS_CM_80060] [SWS_CM_80061] [SWS_CM_80083] [SWS_CM_80092] [SWS_CM_80093]

Requirement	Description	Satisfied by
[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_80071] [SWS_CM_80072] [SWS_CM_80073] [SWS_CM_80074] [SWS_CM_80076] [SWS_CM_80077] [SWS_CM_80078] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80081] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80085] [SWS_CM_80086] [SWS_CM_80087] [SWS_CM_80088] [SWS_CM_80089] [SWS_CM_80090] [SWS_CM_80091] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094] [SWS_CM_80096] [SWS_CM_80097] [SWS_CM_80098] [SWS_CM_80099]
[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_80037] [SWS_CM_80038] [SWS_CM_80047] [SWS_CM_80048] [SWS_CM_80064] [SWS_CM_80079] [SWS_CM_80080] [SWS_CM_80089] [SWS_CM_80090]
[RS_SOMEIP_00012]	SOME/IP protocol shall support session handling	[SWS_CM_10301] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10333] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_80040] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80082] [SWS_CM_80092] [SWS_CM_80093]

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_80027] [SWS_CM_80028] [SWS_CM_80041] [SWS_CM_80052] [SWS_CM_80069] [SWS_CM_80083] [SWS_CM_80093]
[RS_SOMEIP_00017]	SOME/IP protocol shall support grouping events into eventgroups	[SWS_CM_10287] [SWS_CM_10319] [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_80027] [SWS_CM_80028] [SWS_CM_80041] [SWS_CM_80052] [SWS_CM_80069] [SWS_CM_80083] [SWS_CM_80093]
[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] [SWS_CM_80040] [SWS_CM_80041] [SWS_CM_80042] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80053] [SWS_CM_80082] [SWS_CM_80083] [SWS_CM_80084] [SWS_CM_80092] [SWS_CM_80093] [SWS_CM_80094]
[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_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80029] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80071]
[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] [SWS_CM_80040] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80082] [SWS_CM_80092] [SWS_CM_80093]
[RS_SOMEIP_00026]	SOME/IP protocol shall define the endianness of header and payload	[SWS_CM_10013] [SWS_CM_10172] [SWS_CM_80002] [SWS_CM_80003]

Requirement	Description	Satisfied by
[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_80043] [SWS_CM_80056] [SWS_CM_80057] [SWS_CM_80074] [SWS_CM_80085] [SWS_CM_80096] [SWS_CM_80097]
[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_80025] [SWS_CM_80026] [SWS_CM_80040] [SWS_CM_80050] [SWS_CM_80052] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80082] [SWS_CM_80092] [SWS_CM_80093]
[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_01032] [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_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

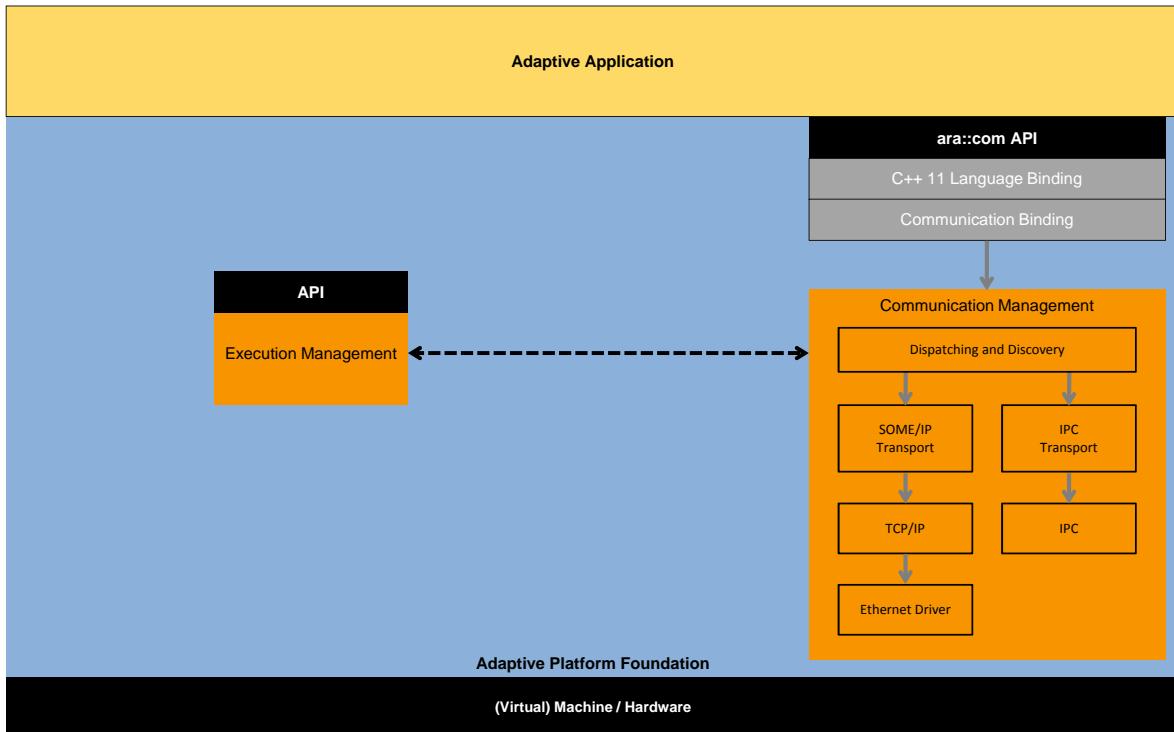


Figure 7.1: Technical Architecture of Communication Management

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 both last-is-best and queued semantics. In case of queued communication, 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 supports filtering of received data.
- 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 [6] and [7].
- Support of Service contract versioning.
- Support Event and Polling style usage of the API equally to enable classic RT style paradigms.
- Fully exploit C++11/14 features in API design to provide usability and comfort for the application developer.

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

7.1.3 Communication paradigms

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.

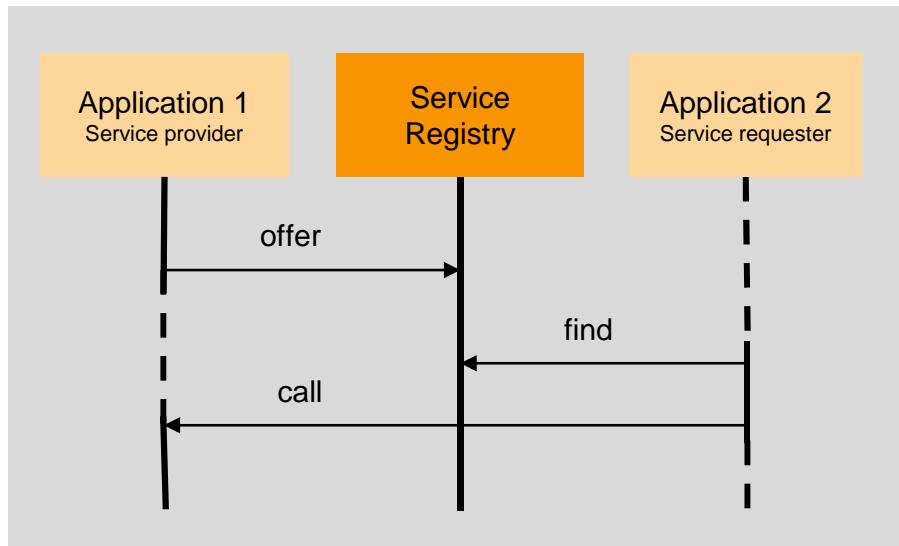


Figure 7.2: Service-Oriented Communication

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.

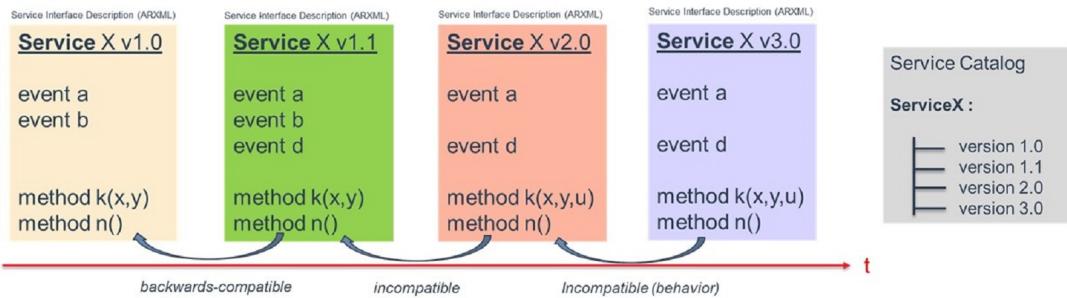


Figure 7.3: Service contract versioning over time

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.

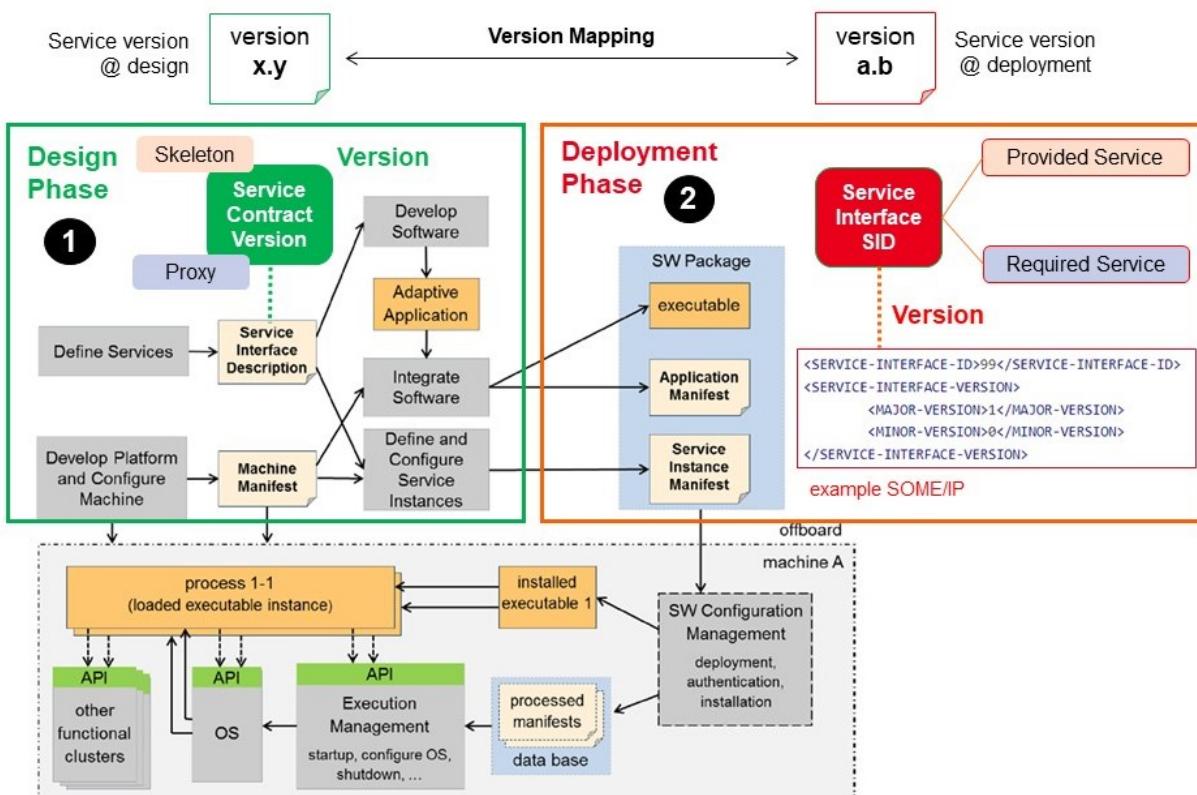


Figure 7.4: Service contract versioning flow

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 is bi-directional that means that there is exactly one version of the `ServiceInterfaceDeployment` mapped to exactly one version of the `ServiceInterface`.

[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 protection in `ara::com` for processing periodic events, that are polled by the `Subscriber`. Note that there are limitations in the released E2E functionality, the limitations are documented in chapter [4.1](#).

[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 `E2E_Protect` and `E2E_Check` - shall comply with the E2E protection protocol as specified in [6] and [7].] ([RS_E2E_08540](#), [RS_CM_00223](#))

7.2.1 Limitations

The specified E2E communication protection for events include only events which are:

- polled for and
- transmitted at least once per fault tolerant time interval.

This means, it requires:

- At consumer side Periodic invocation of the method `GetNewSamples` (see [\[SWS_CM_00701\]](#)) in a polling mode and
- At provider side Periodic or mixed-periodic invocation of the method `Send` (see [\[SWS_CM_00162\]](#) and [\[SWS_CM_90437\]](#)).

If `GetNewSamples` or `Send` is not invoked periodically, then some communication failure modes may not be detected (e.g. loss, delay and possibly also repetition). In this case, additional measures need to be implemented at application level to address

those non-detected failure modes and complete E2E protection or arguments are to be provided showing that these failure modes are not relevant for a particular project.

The values of the following E2E parameters are defined by the standard and shall not be changed (see [7]):

- dataIdMode
- counterOffset
- crcOffset
- dataIdNibbleOffset
- offset

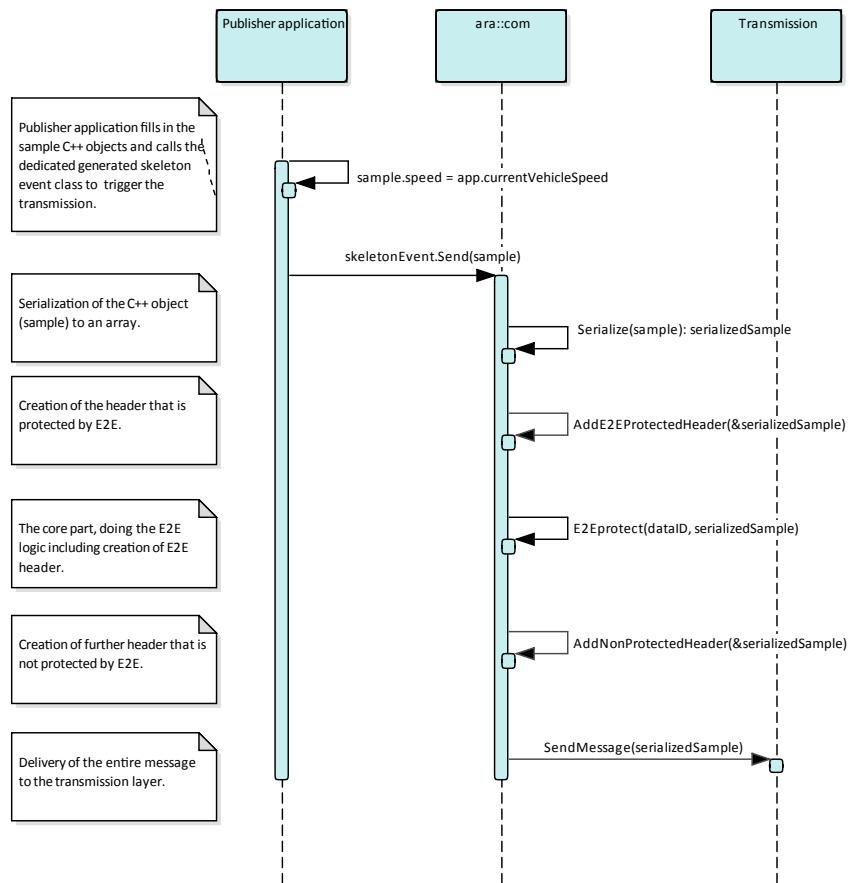
[EndToEndTransformationComSpecProps](#) are not supported.

7.2.2 Publisher

[SWS_CM_90401]{DRAFT} [For e2e-protected events, E2E protection shall be performed within the context of `Send`, by invoking `E2EProtect` on the serialized data according to [RS_CM_00222, RS_E2E_08540, PRS_E2E_USE_00236].] ([RS_E2E_08540](#))

The serialized data include boths a non-protected part as well as athe part to be protected part (see [PRS_E2E_USE_00236]). The data to be protected are the only input to `E2E_protect`.

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


Figure 7.5: E2E Publisher

[SWS_CM_90430]{DRAFT} [For e2e-protected events, Send shall serialize the sample according to the agreed serialization protocol, resulting in serialized Sample.] ([RS_E2E_08540](#))

[SWS_CM_90403]{DRAFT} [For e2e-protected events, the End2EndEventProtectionProps.dataId shall be used.] ([RS_CM_00200](#), [RS_E2E_08540](#))

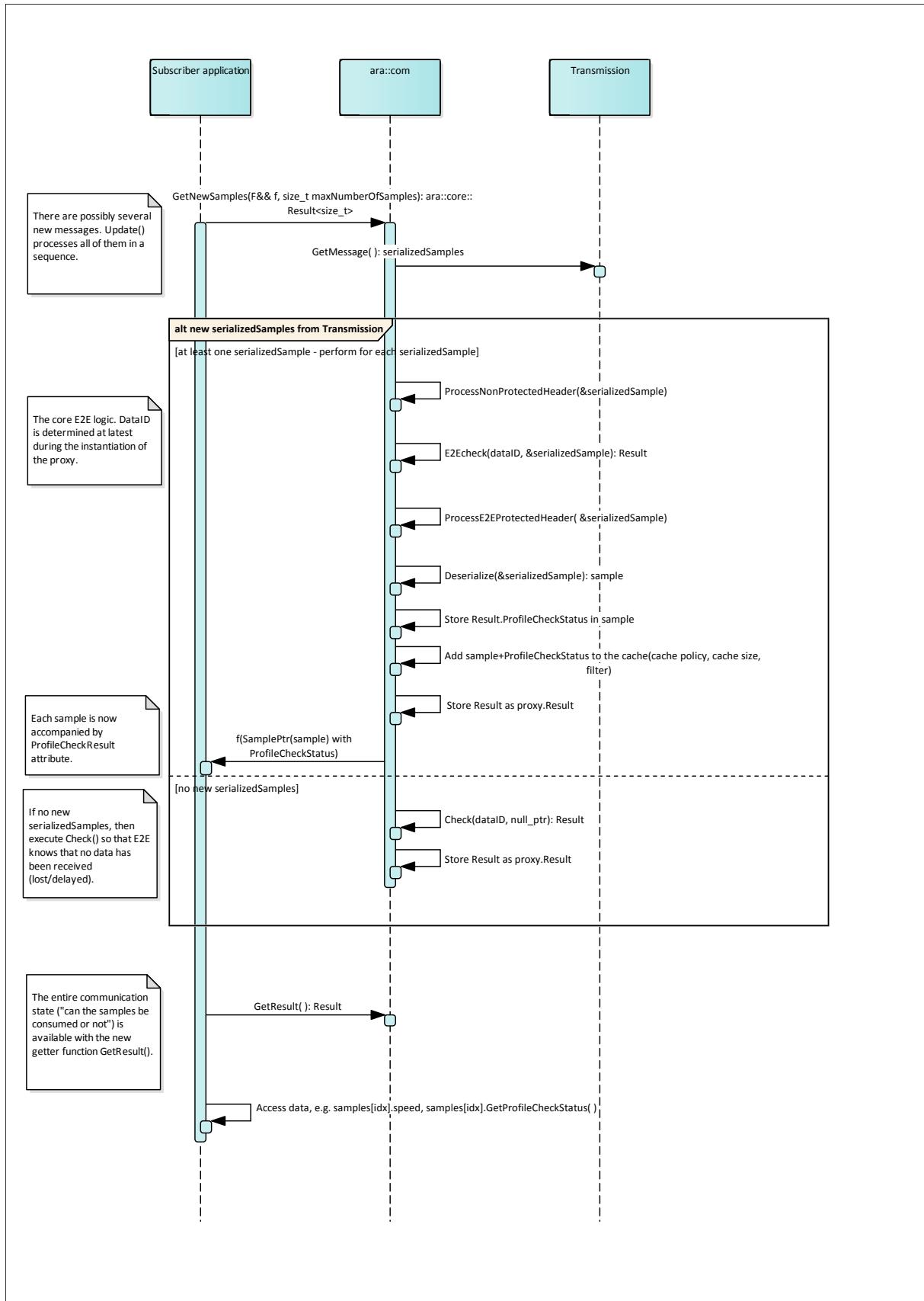
[SWS_CM_90404]{DRAFT} [For e2e-protected events, the e2e protection header shall be added to the message according to [PRS_SOMEIP_00941].] ([RS_E2E_08540](#))

[SWS_CM_90405]{DRAFT} [For e2e-protected events, after the e2e protection is done, Send shall add the non-e2e-protected header (if any) and trigger the transmission.] ([RS_E2E_08540](#))

7.2.3 Subscriber - GetNewSamples

[SWS_CM_90406]{DRAFT} [For e2e-protected events, E2E_check [7] shall be performed within the context of GetNewSamples.] ([RS_E2E_08540](#))

Figure 7.6 shows an overview of the interaction of components involved during the E2E check.


Figure 7.6: E2E Subscriber

[SWS_CM_90407]{DRAFT} [For e2e-protected events, GetNewSamples shall first get the collection of all SerializedSamples that have not been fetched in the last triggering of this GetNewSamples function.] ([RS_E2E_08540](#))

7.2.3.1 Case 1 - there are one or more serialized samples

For each specific Event class belonging to a specific ServiceProxy class the E2E dataID - based on Service ID, Service Instance ID and Event ID - is available.

For e2e-protected events, in case one or more SerializedSamples are received, then for each SerializedSample, the following steps are to be done:

[SWS_CM_90408]{DRAFT} [For the given e2e-protected SerializedSample, GetNewSamples shall process the non-e2e protected header (if any) of the serializedSample.] ([RS_E2E_08540](#))

[SWS_CM_90410]{DRAFT} [For the given e2e-protected SerializedSample, GetNewSamples shall invoke the E2E_check, providing its E2E dataID and serializedSample.] ([RS_E2E_08540](#))

[SWS_CM_90411]{DRAFT} [In return, for the given e2e-protected SerializedSample, E2E_check shall provide Result containing SMState and ProfileCheckStatus.] ([RS_E2E_08540](#), [RS_E2E_08534](#))

[SWS_CM_90412]{DRAFT} [For the given e2e-protected SerializedSample, GetNewSamples shall deserialize it, resulting with deserialized sample.] ([RS_E2E_08540](#))

[SWS_CM_90413]{DRAFT} [For the given e2e-protected SerializedSample, GetNewSamples shall store the ProfileCheckStatus in the SamplePtr and it shall update/overwrite the global SMState within its specific Event class.] ([RS_E2E_08540](#), [RS_E2E_08534](#))

7.2.3.2 Case 2 - there are no serialized samples

For each specific Event class belonging to a specific ServiceProxy class the E2E dataID - based on Service ID, Service Instance ID and Event ID - is available.

In case no e2e-protected SerializedSamples are received, the steps are simpler and E2E works as timeout detection.

[SWS_CM_90415]{DRAFT} [In case no e2e-protected SerializedSamples are received, GetNewSamples shall invoke the E2E_check, providing its E2E dataID and a null sample.] ([RS_E2E_08540](#))

[SWS_CM_90416]{DRAFT} [In case no e2e-protected SerializedSamples are received, in return, E2E_check shall provide Result containing SMState and ProfileCheckStatus.] ([RS_E2E_08540](#), [RS_E2E_08534](#))

[SWS_CM_90417]{DRAFT} [In case no e2e-protected SerializedSamples are received, GetNewSamples shall update/overwrite the global SMState within its specific Event class.] ([RS_CM_00222](#), [RS_E2E_08540](#), [RS_E2E_08534](#))

7.2.4 Subscriber - Callback f

The user provided Callable f, which will be invoked for each received sample contains the dependent ProfileCheckStatus.

7.2.5 Subscriber - Access to E2E information

[SWS_CM_10475]{DRAFT} [Each Event shall have a getter function GetSMState allowing to accessInside of a specific Event class belonging to the specific ServiceProxy class, a GetSMState method shall be provided (see [\[SWS_CM_00701\]](#)).] ([RS_CM_00225](#), [RS_E2E_08534](#))

Each SamplePtr provides a getter function GetProfileCheckStatus to access the ProfileCheckStatus of each Sample (see [\[SWS_CM_90420\]](#)).

[SWS_CM_90431]{DRAFT} [The GetSMState method shall provide access to the global SMState, which was determined by the last run of E2E_check function invoked during the last call of GetNewSamples (see [\[SWS_CM_90431\]](#)).] ([RS_CM_00225](#), [RS_E2E_08534](#))

1 ara::com::e2e::SMState GetSMState() const noexcept;

7.3 End-to-end communication protection for Methods

This section specifies the integration of E2E protection in ara::com for processing methods. This includes E2E protection for a method REQUEST as well as E2E protection for any response (ERROR or normal RESPONSE).

[SWS_CM_10460]{DRAFT} [An e2e-protected method shall have its options configured in [End2EndMethodProtectionProps](#) and [E2EProfileConfiguration](#).] ([RS_CM_00222](#), [RS_E2E_08540](#))

[SWS_CM_10461]{DRAFT} [The E2E functions mentioned in this section **E2E_protect** and **E2E_check** - shall comply with the E2E protection protocol as specified in [6] and [7].] ([RS_CM_00222](#), [RS_E2E_08540](#), [RS_CM_00223](#))

7.3.1 Limitations

The specified E2E communication protection for methods is limited to

- Communication between N clients and one dedicated server (N:1); this means, an E2E protected service is provided by exactly one server per system.
- Communication errors which are detected at **lower layers** (e.g., by Ethernet FCS, IP header checksum, UDP checksum, SOME/IP header irregularities) will lead to **discarding the related message**. Thus, these messages might not reach the adaptive application process at all.
- The processing mode kEvent (concurrent threads) is not supported for E2E protected methods.

The specified E2E communication protection for methods may not detect all communication failure modes:

- If one of the method requests is not invoked periodically, then some communication failure modes (loss, delay) may not be detected.
- As the defined E2E protection mechanisms assume no a priori knowledge at the server about the client that is just requesting data (N:1 communication), communication failure modes (repetition, insertion) which are client specific may not be detected.

In this case, additional measures need to be implemented at application level to address those non-detected failure modes and complete E2E protection or arguments are to be provided showing that these failure modes are not relevant for a particular project.

The values of the following E2E parameters are defined by the standard and shall not be changed. See [7].

- dataIdMode
- counterOffset
- crcOffset
- dataIdNibbleOffset
- offset

[EndToEndTransformationComSpecProps](#) are not supported.

7.3.2 Call a service method (Client)

[SWS_CM_10462]{DRAFT} [For e2e-protected methods, E2E protection shall be performed within the context of the call of the service method, by invoking E2E_protect on the serialized data according to PRS_E2E_USE_00236.] ([RS_CM_00222](#), [RS_E2E_08540](#))

Note: See also C++ API reference chapter 8.

The serialized data include both a non-protected part as well as a part to be protected (see PRS_E2E_USE_00236). The data to be protected are the only input to E2E_protect.

Figure 7.7 shows an overview of the interaction of components involved during the E2E protection.

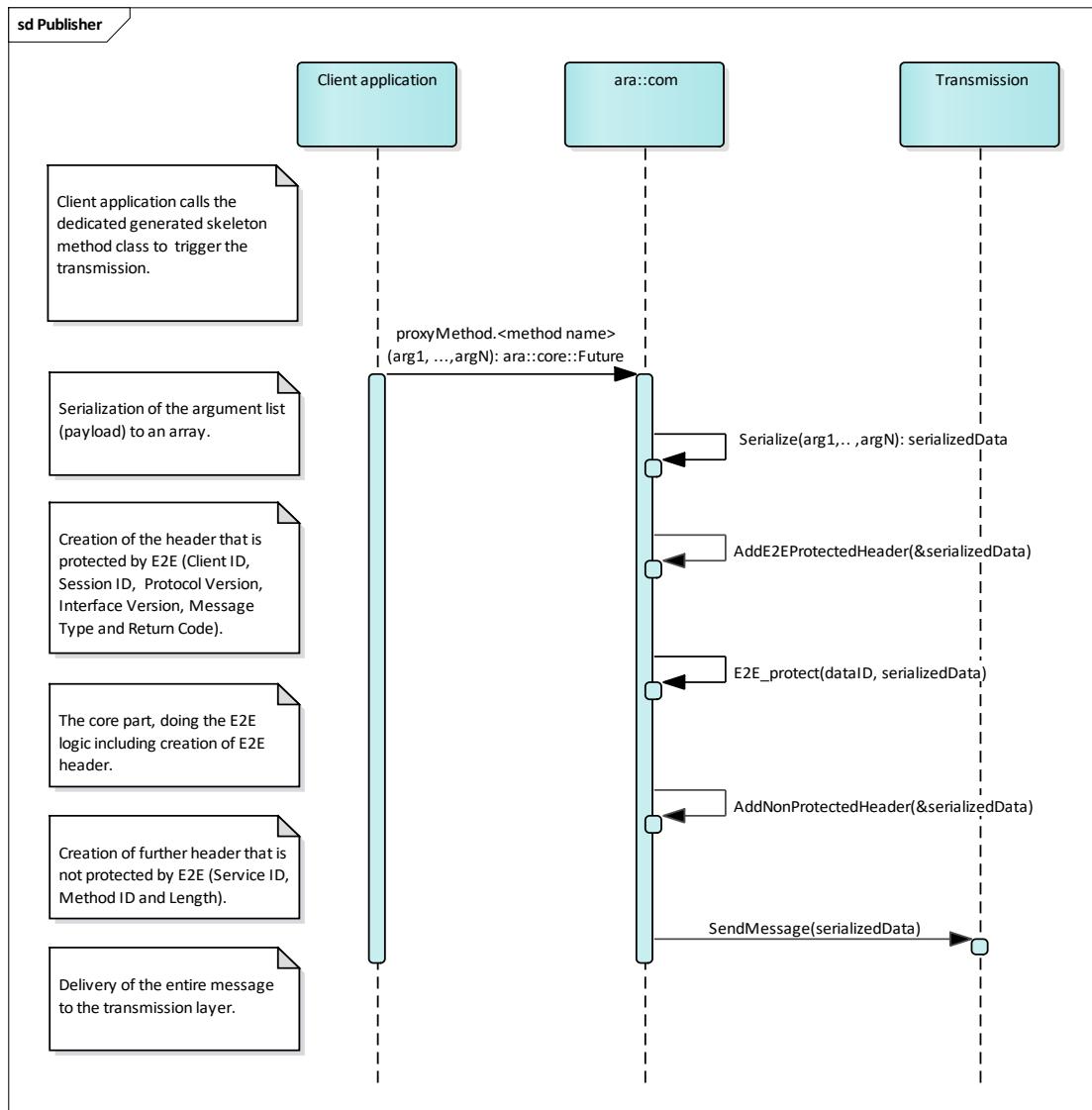


Figure 7.7: Interaction of components during E2E protection

[SWS_CM_10463]{DRAFT} [For e2e-protected methods, the End2EndMethodProtectionProps.dataID shall be used.]([RS_CM_00222](#), [RS_CM_00200](#), [RS_E2E_08540](#))

The E2E dataID could be derived as a unique number based on uniquely available ids like Service ID, Instance ID and Method ID.

[SWS_CM_10464]{DRAFT} [For e2e-protected methods, the e2e protection header shall be added to the request message according to PRS_SOMEIP_00941.]([RS_CM_00222](#), [RS_E2E_08540](#))

[SWS_CM_10465]{DRAFT} [For e2e-protected methods, the response message has the message counter value of the request message. In case the message counter is different, the response message shall be discarded (without any further processing).] ([RS_CM_00222](#), [RS_E2E_08540](#))

Implementation Hint: The counter can be extracted from the result of the E2E_PXX_Check() function.

ara::com does not support any timeout supervision. 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 according mechanisms, e.g., by using the wait_for(), wait_until(), or the is_ready() methods of the ara::core::Future.

7.3.3 Processing of service methods (Server)

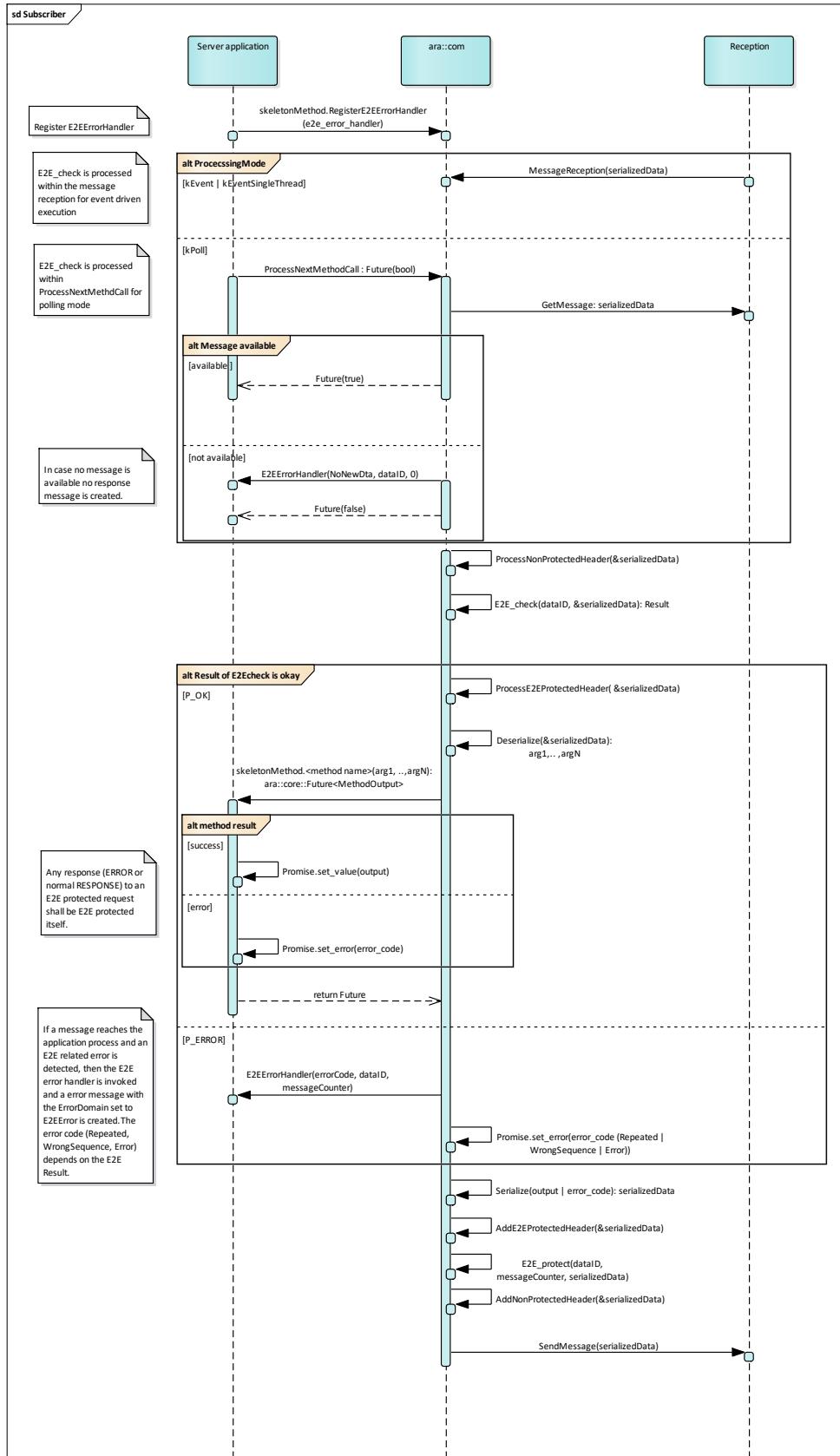
[SWS_CM_10466]{DRAFT} [For e2e-protected methods, E2ECheck [7] shall be performed within the context of the message reception within the ServiceSkeleton if the MethodCallProcessingMode is set to kEventSingleThread.]([RS_CM_00222](#), [RS_E2E_08540](#))

[SWS_CM_10467]{DRAFT} [In case a MethodCallProcessingMode of kEvent has been passed to the constructor of the ServiceSkeleton (see [[SWS_CM_00130](#)] or [[SWS_CM_10435](#)]), the constructor shall not succeed.]()

Note: A MethodCallProcessingMode set to kEvent is not supported for e2e-protected methods.

[SWS_CM_10468]{DRAFT} [For e2e-protected methods, E2ECheck [7] shall be performed after the call of ProcessNextMethodCall within the ServiceSkeleton if the MethodCallProcessingMode is set to kPoll and a pending request message is available.]([RS_CM_00222](#), [RS_E2E_08540](#))

Figure 7.8 shows an overview of the interaction of components involved during the E2E check.


Figure 7.8: Interaction of components during E2E check

[SWS_CM_10469]{DRAFT} [For e2e-protected methods, E2E_protect [7] shall be called with the received Data ID and message counter.]
|(RS_CM_00222, RS_E2E-08540)

Note: Any response message has the same Data ID and message counter as the request message to simplify the multiple client scenarios and allow the client to monitor the message counter.

7.3.4 E2E error handler

[SWS_CM_10470]{DRAFT} [E2E Error Handler d The Communication Management shall provide the definition of E2EEErrorHandler as a function wrapper with parameters for ErrorCode, DataID and MessageCounter for the handler function. The handler gets called within an separated thread by the Communication Management software in case E2ECheck reports an E2E error. The server application has to 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++ generalpurpose polymorphic function wrapper std::function, but this is not mandatory and is allowed to be changed by the Communication Management software provider.

```
1 using E2EEErrorHandler = std::function<void(aralcom::e2e::E2EEErrorCode  
errorCode, aralcom::e2e::DataID dataID, aralcom::e2e::MessageCounter  
messageCounter)>;
```

|(RS_CM_00203)

The server application can get notified for E2E errors in request messages for methods.

[SWS_CM_10471]{DRAFT} **E2E Error Handler** [The E2EEErrorHandler shall be called with the parameters set to errorCode = kNoNewData, dataID = Data ID, messageCounter = 0; if within the call of ProcessNextMethodCall no new request message is available. In case there was no message reception at all, the parameter errorCode is set to kNotAvailable.]()

[SWS_CM_10472]{DRAFT} **E2E Error Response** [In case E2ECheck reports an E2E error, the further reception steps are not processed. Further a response message containing an error structured according to [SWS_CM_10474] shall be constructed. The *ErrorDomain* shall be set to *E2EEError* and the *ErrorCode* shall be set to the corresponding error value of E2E_check according to [SWS_CM_90421].]()

[SWS_CM_10473]{DRAFT} **E2E Error Response** [The client shall map the error message [SWS_CM_10472] to the error domain aralcom::E2EEError.]()

7.4 Raw Data Streaming

7.4.1 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 is 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 configurable by setting attributes in [RawDataStreamMethodDeployment](#) in the deployment model of the Raw Data Stream Interface. The attributes that can be configured is described in [TPS_ManifestSpecification](#) [5]. 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](#) [5].

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

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 must 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.

Figure 7.9 shows the logical view of the usage of RawDataStream instances.

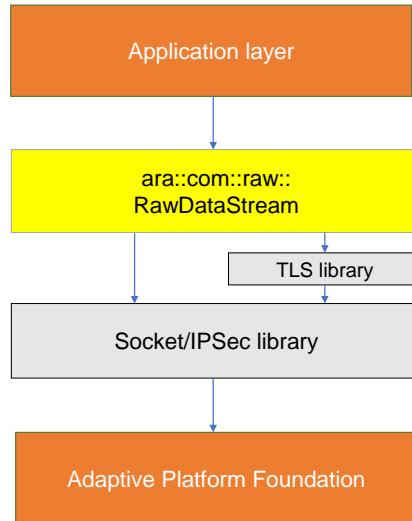


Figure 7.9: Raw Data Stream Logical View.

7.4.1.1 Limitations

Currently, only the client side of communication channels is defined in the interface. In future releases, the possibility to define a server side solution for Raw Data Streams shall be provided.

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

7.4.2 Raw Data Streaming

For the Raw Data Stream C++ API reference, see chapter 8.1.3.19.

[SWS_CM_10476]{DRAFT} Defining a RawDataStream [The Raw Data Streaming Interface shall be implemented as a class, `RawDataStream`, with each instance handling one stream. Currently, one socket shall be defined for each stream instance.]
([RS_CM_00410](#), [RS_CM_00411](#))

Four operations for handling the streams shall be available for each `RawDataStream` instance:

[SWS_CM_10477]{DRAFT} Connect stream link [Shall initialize the socket, establish the communication link (socket connection), and if configured, a secure channel

for the stream. The operation shall return success or failure of the connection. If not successful an error code shall indicate the reason.

If a timeout value > 0 is set for the Connect operation, it shall return with a timeout error if the connection time exceeds the timeout value. Timeout value 0 is not applicable to this operation.] ([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

[SWS_CM_10478]{DRAFT} Shutdown stream link [Shall destroy the communication link for the stream. The operation shall return success or failure. If not successful an error code shall indicate the reason.

If a timeout value > 0 is set for the Shutdown operation, it shall return with a timeout error if the operation time exceeds the timeout value. Timeout value 0 is not applicable to this operation.] ([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

[SWS_CM_10479]{DRAFT} Read data from stream [Shall read data, a sequence of bytes, from the stream and move it to a buffer returned as result from the function, together with the actual number of bytes transferred. The requested number of bytes shall be provided in the call.

- If no timeout is set for the read operation it shall block until data becomes available. Then it returns the data that is available and the number of bytes read.
- If a timeout value > 0 is set for the read operation, it shall return with a timeout error if no data is made available during for the timeout period, otherwise it returns the data that is available and the number of bytes read.
- If a timeout value = 0 is set, it shall return the data that is available at the call and return immediately and the number of bytes read. If no data is available, 0 bytes should be returned.

] ([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

[SWS_CM_10480]{DRAFT} Write data to stream [Shall write data, a sequence of bytes, to the stream and send it out on the socket connection. The actual number of bytes transferred shall be returned.

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.

If a timeout value > 0 is set for the Write operation, it shall return with a timeout error if the operation time exceeds the timeout value. Timeout value 0 is not applicable to this operation.] ([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

7.5 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_00207](#), [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.

7.5.1 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] [The SOME/IP network binding shall implement the SOME/IP Protocol and the SOME/IP Service Discovery Protocol defined in [4] and [12].] ([RS_CM_00204](#), [RS_CM_00205](#))

[SWS_CM_10013] [All headers shall be encoded in network byte order Big Endian (MostSignificantByteFirst) [RFC 791].] ([RS_CM_00204](#), [RS_SOMEIP_00026](#))

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

[SWS_CM_10172] [The byte order of the parameters inside the payload shall be defined by `byteOrder` of `ApSomeipTransformationProps`.] ([RS_CM_00204](#), [RS_SOMEIP_00026](#))

7.5.1.1 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](#))

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 (INITIAL_DELAY_MIN, _MAX) followed by Repetition

Wait phase (REPETITIONS_BASE_DELAY, REPETITIONS_MAX) and main phase (CYCLIC_OFFER_DELAY).

(See also [PRS_SOMEIPSD_00395], [PRS_SOMEIPSD_00397], [PRS_SOMEIPSD_00399], [PRS_SOMEIPSD_00416], [PRS_SOMEIPSD_00435] and [PRS_SOMEIPSD_00752]) (RS_CM_00204, RS_CM_00102, RS_SOMEIPSD_00024)

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 `Initials-dDelayConfig` 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 (0x00).
- 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 "ANY" 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 "ANY", then 0xFFFF FFFF shall be used.

- TTL shall be derived from the Manifest where the `SomeipSdClientService-InstanceConfig` 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](#))

[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 (0x01).
- 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 `SomeipSdServerService-InstanceConfig` 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 configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.

- 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](#))

[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 (0x01).
- 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](#))

[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](#))

[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](#))

[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](#) (0x06).
- 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 [eventId](#) 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 `Network-Endpoint` in the role `unicastNetworkEndpoint` where an IPv4 Address is configured in the `Ipv4Configuration` element.
- IPv6 Endpoint Option shall be sent if the offer message contains an IPv6 Endpoint Option. In this case the IPv6 Address sent in the IPv6 Endpoint Option of the SubscribeEventgroup message is configured in the Manifest where the `RequiredSomeipServiceInstance` element is mapped with the `ServiceInstanceToMachineMapping` to an `EthernetCommunicationConnector` of a `Machine`. The `EthernetCommunicationConnector` refers to a `Network-Endpoint` in the role `unicastNetworkEndpoint` where an IPv6 Address is configured in the `Ipv6Configuration` element.
- The Transport Layer Protocol used in the IPv4 Endpoint option and/or IPv6 Endpoint option shall be derived from the Manifest where the `SomeipEventGroup` points either to `SomeipEventDeployments` where the `transportProtocol` is set to `udp` or to `tcp`. The `SomeipServiceInstanceToMachineMapping` element that maps the `RequiredSomeipServiceInstance` to an `EthernetCommunicationConnector` of a `Machine` the transport protocol and the port number.
 - 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 configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.
 - 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 configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.

] (RS_CM_00204, RS_CM_00200, RS_CM_00103, RS_SOMEIPSD_00006)

[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 (0x07).
- Serviceld shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- Instanceld 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*)

[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 (0x07).
- Serviceld shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- Instanceld 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*)

[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 Events 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*)

[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 (0x06).
- 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*)

7.5.1.2 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 `unicast-NetworkEndpoint` 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 `ProvidedServiceInstance` for transmission over UDP [For a `ProvidedServiceInstance` all `method` responses and `events` 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 that 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].

7.5.1.3 Execution context of message reception actions

In the following sections the term "upon reception" will be used to describe that 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 separate execution context to leave room for potential optimizations of a concrete ara::com implementation.

The only restriction imposed here refers to the execution context of the `EventReceiveHandler` (see [SWS_CM_00309]). – Executing the `EventReceiveHandler` in the context of the `GetNewSamples` (see [SWS_CM_00701]) method is not allowed, since according to [SWS_CM_00181] the `EventReceiveHandler` shall use the `GetNewSamples` method to access the retrieved event data.

7.5.1.4 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 `event-Group` 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 the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x000 (see [PRS_SOMEIP_00932] and [PRS_SOMEIP_00925]). In case of active Session Handling the Session ID is used for event messages and thus shall be incremented (with proper wrap around) upon every transmission of an event message

(see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).

- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to NOTIFICATION (0x02).
- The Return Code (see [PRS_SOMEIP_00040]) is unused for event messages and thus (according to [PRS_SOMEIP_00040]) shall be set to E_OK (0x00).
- The Payload shall contain the serialized payload (i.e., the serialized `Variable-DataPrototype` 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 [7.5.1.7](#).

[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.
- Verify that the Length (see [PRS_SOMEIP_00042]) is larger than 7.
- Use the Message Type (see [PRS_SOMEIP_00055]) which is set to NOTIFICATION (0x02) to determine that the received SOME/IP message is actually a SOME/IP event messages.
- 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_00040]) is set to E_OK (0x00).

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_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 `SubscribeEvent-group` 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 7.5.1.7.

[SWS_CM_10295] Providing the received event data [The serialized 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*)

7.5.1.5 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 `SomeipServiceInstance-ToMachineMapping` 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 achieved by dynamically generating unique client IDs upon construction of the `ServiceProxy`.
- The Session ID (see [PRS_SOMEIP_00703]) shall be set to 0x0001 for the first call of a particular method by a given client and shall be incremented by 1 after each call performed by this client for the respective method (see [PRS_SOMEIP_00533]). Once the Session ID reaches 0xFFFF, it shall wrap around and start with 0x0001 again (see [PRS_SOMEIP_00521]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to REQUEST_NO_RETURN (0x01) in case the `ClientServerOperation` referenced by `methodDeployment.method` contains a `fireAndForget` attribute which is set to `true`. The Message Type shall be set to REQUEST (0x00) otherwise.
- The Return Code (see [PRS_SOMEIP_00040]) 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 7.5.1.7.

[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 `SomeipMethodDeployments` of the `SomeipServiceInterfaceDeployment`.
- Verify that the Message Type (see [PRS_SOMEIP_00055]) is set to REQUEST_NO_RETURN (0x01) in case the the `ClientServerOperation` referenced by `methodDeployment.method` of the `SomeipMethodDeployment` with matching `methodId` attribute contains a `fireAndForget` attribute which is set to `true`. Verify that the Message Type is set to REQUEST (0x00) otherwise.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches `SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion`.
- Verify that the Return Code (see [PRS_SOMEIP_00040]) 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 `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_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 [7.5.1.7](#).

[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_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 `ApApplicationErrors` referenced by the `ClientServerOperation` in the role `possibleApError` 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 `possibleApError1`. The Message Type shall be set to `RESPONSE` (0x80) otherwise.
- The Return Code (see [PRS_SOMEIP_00040]) 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`. 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

¹Note 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 `ApplicationErrors` 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.

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 7.5.1.7.

[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
- 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_00040]) 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

²The additional case of SOME/IP response messages with a Return Code (see [PRS_SOMEIP_00040]) 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 `ApplicationErrors` are raised.

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 7.5.1.7.

[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_00040]) contains a value larger than 0x1F the corresponding value of the `ApApplicationError.errorCode` attribute shall be determined by subtracting 0x1F from the Return Code value. Using this computed `ApApplicationError.errorCode` attribute value and the `ApApplicationError.errorCode` attribute of all `ApApplicationErrors` referenced in role `possibleApError` by the `ClientServerOperation` corresponding to the method determined according to [SWS_CM_10314], the right application error shall be identified.

If this computed `ApApplicationError.errorCode` attribute value does not match any of the `ApApplicationError.errorCode` attributes of all `ApApplicationErrors` referenced in role `possibleApError` by the `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].

If this computed `ApApplicationError.errorCode` attribute value does match more than one of the `ApApplicationError.errorCode` attributes of all `ApApplicationErrors` referenced in role `possibleApError` by the `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)

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_00040]) contains a value equal to 0x01 (E_NOT_OK) then the corresponding [ApApplicationError](#) shall be identified 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 RESPONSE (0x80) [If the Return Code see [PRS_SOMEIP_00040]) 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](#) 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 serialized payload according to [SWS_CM_10316] as an argument. The `SetError` operation shall be invoked in case of a received error response message using the determined application error according to [SWS_CM_10358] and [SWS_CM_10429] of type `ara::core::ErrorCode` as an argument.] ([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](#))

7.5.1.6 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 be 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 the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x000 (see [PRS_SOMEIP_00932] and [PRS_SOMEIP_00925]). In case of active Session Handling the Session ID is used for event messages and thus shall be incremented (with proper wrap around) upon every transmission of an event message (see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to NOTIFICATION (0x02).
- The Return Code (see [PRS_SOMEIP_00040]) is unused for event messages and thus (according to [PRS_SOMEIP_00040]) 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 [7.5.1.7](#).

[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 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.notifiers` 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 `SubscribeEvent`-

group 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 7.5.1.7.

[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_00040]) 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 [7.5.1.7](#).

[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_00040](#)]) 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_00200](#),

[00212](#), [RS_CM_00213](#), [RS_SOMEIP_00007](#), [RS_SOMEIP_00009](#), [RS_SOMEIP_00028](#)) The SOME/IP serialization rules are explained in section [7.5.1.7](#).

[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_00040]) 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 7.5.1.7.

[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 7.5.1.7.

[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 serialized 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](#))

7.5.1.7 Serialization of Payload

[SWS_CM_10034] [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] [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] 「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 7.5.1.7.4).

[SWS_CM_10260] 「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_00201](#), [RS_CM_00202](#), [RS_CM_00211](#)) (see chapter 7.5.1.7.4)

[SWS_CM_11262] 「If `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` is not set for a variable data length data element, the value of `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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 7.5.1.7.4)

[SWS_CM_11263] 「If `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` and `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] 「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 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] [If more data than expected shall be serialized, the unexpected data shall be discarded. The known fraction shall be considered.] ([RS_CM_00204](#), [RS_CM_00202](#))

[SWS_CM_10017] [If less data than expected shall be serialized and the data to be serialized 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.

7.5.1.7.1 Basic Data Types

[SWS_CM_10036] [The primitive `StdCppImplementationDataTypes` 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)
uint8_t	unsigned Integer	8	
uint16_t	unsigned Integer	16	
uint32_t	unsigned Integer	32	
uint64_t	unsigned Integer	64	
int8_t	signed Integer	8	
int16_t	signed Integer	16	
int32_t	signed Integer	32	
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 `StdCppImplementationDataTypes` supported for serialization

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

7.5.1.7.2 Enumeration Data Types

[SWS_CM_10361] [`Enumeration Data Types` 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 [\[SWS_CM_00424\]](#))] ([RS_CM_00204](#), [RS_CM_00201](#), [RS_CM_00202](#), [RS_CM_00211](#))

7.5.1.7.3 Scale Linear And Texttable Data Types

[SWS_CM_10391] [Scale Linear And Texttable Data Types shall be serialized according to [SWS_CM_10361] based on the Enumeration Data Type they were specified with (see [SWS_CM_10409]).] (*RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211*)

7.5.1.7.4 Structured Data Types (structs)

[SWS_CM_10042] [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 uint8_t and a 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 uint32_t; therefore, the 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] [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] [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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.byteOrder` is set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is not set, the attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If `SomeipDataPrototypeTransformationProps.someip-TransformationProps.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] [If `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.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_10218] [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] [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.

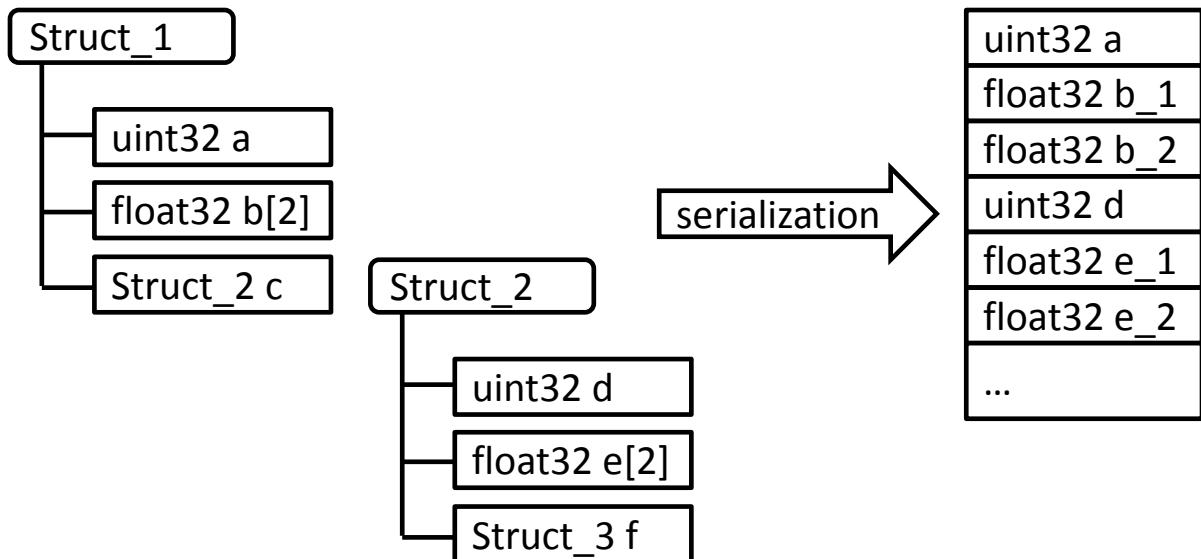


Figure 7.10: Serialization of Structs without Length Fields (Example)

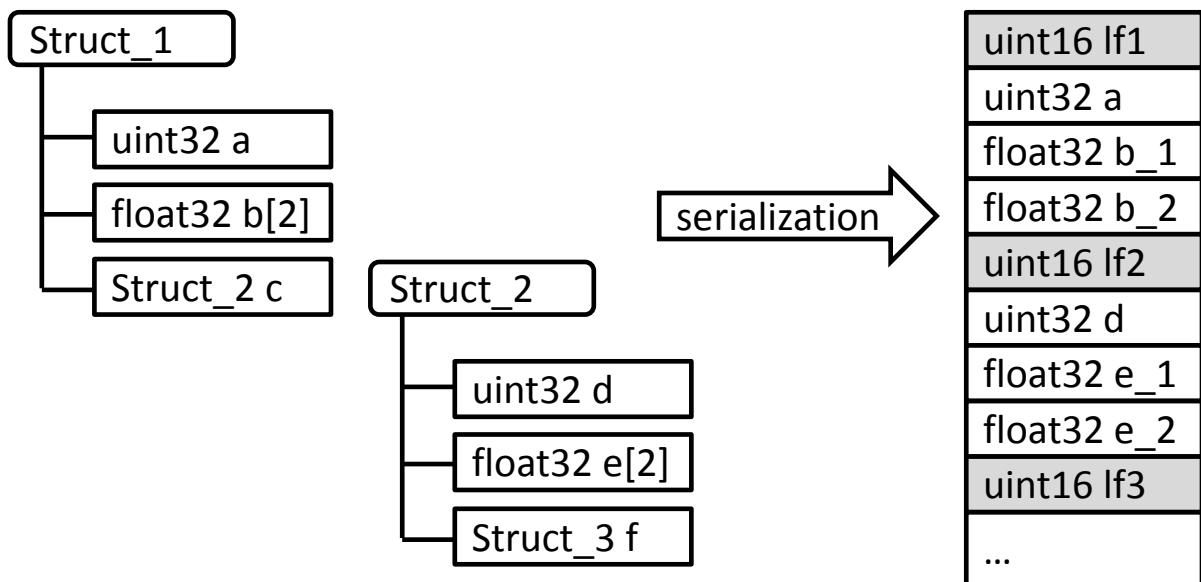


Figure 7.11: Serialization of Structs with Length Fields (Example)

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

7.5.1.7.5 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 member-

s/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 `CppImplementation-DataTypeElement`s. Method arguments are represented by `ArgumentDataPrototypes`.

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

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

Whether an optional member is actually present in the struct or not, must be determined during runtime. This is realized in the Adaptive Platform using the `ara::core::Optional` class template (see [8.1.2.4.2 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 [4].

[SWS_CM_90443] [If `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.isDynamicLengthField-Size` 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 `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.sizeOfStructLengthField`, `sizeOfArrayLengthField` or `sizeOfStringLengthField`.] ([RS_CM_00204](#))

[SWS_CM_90444] [If `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.isDynamicLengthField-Size` 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 independent of the setting of `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.isDynamicLengthFieldSize`.] ([RS_CM_00204](#))

[SWS_CM_90446] [If a Data ID is defined for an `ArgumentDataPrototype` or `CppImplementationDataType` by means of `TransformationPropsToServiceInterfaceElementMappingSet.TlvDataIdDefinition.id`, a tag shall be inserted in the serialized byte stream.] ([RS_CM_00204](#))

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

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

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

[SWS_CM_90451] [TransformationPropsToServiceInterfaceElementMap-pingSet.mapping.transformationProps.byteOrder shall define the byte order for the length field.] ([RS_CM_00204](#))

[SWS_CM_90452] [TransformationPropsToServiceInterfaceElementMap-pingSet.mapping.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 [4].

7.5.1.7.6 Strings

[SWS_CM_10053] [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] [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 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 shall have an even length.] ([RS_CM_00204](#), [RS_CM_00201](#), [RS_CM_00202](#), [RS_CM_00211](#))

[SWS_CM_10057] [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] [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] [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] [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] [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] [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 precedes the actual string data. The value of this length field holds the length of the string including the BOM and any string termination in units of bytes.

[SWS_CM_10271] [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] [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 `SomeipDataProto-`

typeTransformationProps.someipTransformationProps.] (RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)

[SWS_CM_10273] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.byteOrder is set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is not set, the attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [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] [If `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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, error handling shall be performed according to [SWS_CORE_00001].
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, error handling shall be performed according to [SWS_CORE_00001].
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](#))

7.5.1.7.7 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] [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] [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] [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 `SomeipDataProto-`

typeTransformationProps.someipTransformationProps.] (*RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211*)

[SWS_CM_00258] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.byteOrder is set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is not set, the attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [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] [If `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [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] [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] [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 `CppClassImplementationDataType` 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] [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 [7.12](#) and Figure [7.13](#) 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.

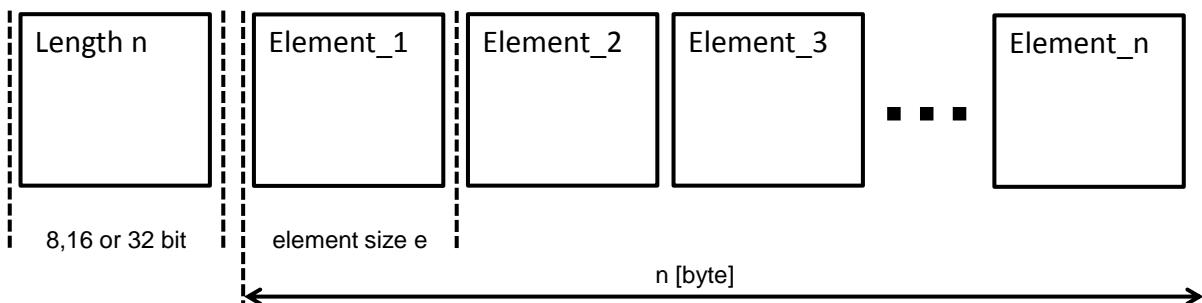


Figure 7.12: One-dimensional arrays (Example)

[SWS_CM_10070] [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] [The serialization of multi-dimensional arrays shall happen in depth-first order.] ([RS_CM_00204](#), [RS_CM_00201](#), [RS_CM_00202](#), [RS_CM_00211](#))

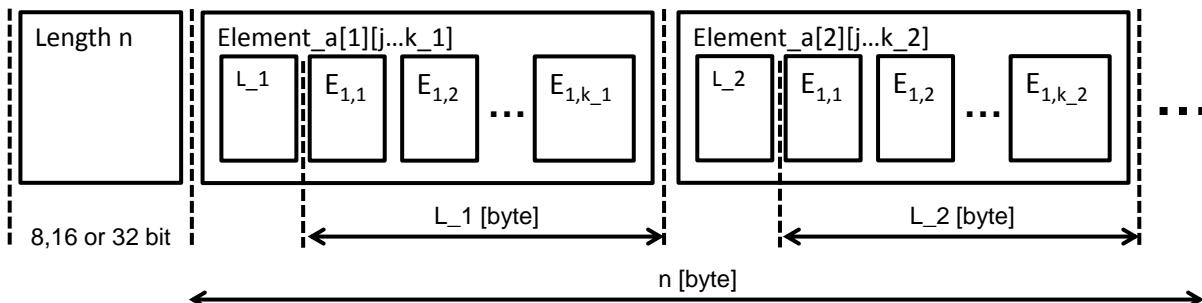


Figure 7.13: Multi-dimensional arrays (Example)

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 [7.13](#).

7.5.1.7.8 Associative Maps

Associative map is modeled as `StdCppImplementationDataType` with `category` ASSOCIATIVE_MAP in the Manifest. As stated in the AUTOSAR Manifest Specification [5] the “natural” language binding in C++ for an associative map is `ara::core::Map<key_type,value_type>` where `key_type` is the data type used for the key of a map element and `value_type` is the data type for the value of a map element. Hereby `key_type` and `value_type` are derived from defined `CppClassTemplateArguments`s aggregated by the `Associative Map Cpp Implementation Data Type`. Please see [SWS_CM_00409] 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] [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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.byteOrder` is set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is not set, the attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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 `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If attribute `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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] [If `TransformationPropsToServiceInterfaceElementMappingSet.mapping.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 7.5.1.7.4)

Note: Adhering to [\[SWS_CM_10266\]](#) is essential to ensure interoperability with the AUTOSAR classic platform where maps may be modelled as `ApplicationArrayType` with a `dynamicArraySizeProfile` of `VSA_LINEAR` where each array element is an `ApplicationRecordDataType` of variable length and thus [\[TPS_SYST_02126\]](#) applies to the individual `ApplicationRecordElements`.

7.5.1.7.9 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] [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] [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] [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] [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 [7.5.1.7.4](#))

[SWS_CM_10227] [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] [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] [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 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] [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](#))

7.5.1.7.9.1 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

7.5.1.7.10 Segmentation of SOME/IP messages

[SWS_CM_10454] [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] [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] [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] [For messages that would fit into one segment no segmentation (i.e. no TP-Header) shall be applied.] ([RS_SOMEIP_00051](#))

7.5.1.8 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 [ProvidedServiceInstance](#) 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](#))

7.5.2 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.

The signal-based network binding is currently a specialization of the SOME/IP network binding and many aspects of the SOME/IP network binding are re-used. The intent is to keep the individual network binding specification parts independent of each other (to provide separate specification documents at a later point in time). The commonalities and differences between the signal-based network binding and the SOME/IP network binding are summarized in table 7.4.

Signal-based binding	SOME/IP binding	Relation category
[SWS_CM_80001]	[SWS_CM_10000]	identical except for serialization
[SWS_CM_80002]	[SWS_CM_10013]	identical
[SWS_CM_80003]	[SWS_CM_10172]	restricted to SOME/IP serialization
[SWS_CM_80004]	-	restricted to signal-based serialization
[SWS_CM_80005]	[SWS_CM_00201]	identical
[SWS_CM_80006]	[SWS_CM_00209]	identical
[SWS_CM_80007]	[SWS_CM_00202]	identical
[SWS_CM_80008]	[SWS_CM_00203]	identical
[SWS_CM_80009]	[SWS_CM_00204]	identical
[SWS_CM_80010]	[SWS_CM_10377]	identical
[SWS_CM_80011]	[SWS_CM_10381]	identical
[SWS_CM_80012]	[SWS_CM_00205]	identical
[SWS_CM_80013]	[SWS_CM_00206]	identical
[SWS_CM_80014]	[SWS_CM_00208]	identical
[SWS_CM_80015]	[SWS_CM_10378]	identical
[SWS_CM_80016]	[SWS_CM_00207]	identical
[SWS_CM_80017]	[SWS_CM_10387]	generalized
[SWS_CM_80018]	[SWS_CM_10388]	identical
[SWS_CM_80019]	[SWS_CM_10389]	generalized
[SWS_CM_80020]	[SWS_CM_10390]	generalized
[SWS_CM_80021]	[SWS_CM_10287]	generalized
[SWS_CM_80022]	[SWS_CM_10288]	generalized





Signal-based binding	SOME/IP binding	Relation category
[SWS_CM_80023]	[SWS_CM_10289]	generalized
[SWS_CM_80024]	[SWS_CM_10290]	generalized
[SWS_CM_80025]	[SWS_CM_10291]	restricted to SOME/IP serialization
[SWS_CM_80026]	-	restricted to signal-based serialization
[SWS_CM_80027]	[SWS_CM_10292]	restricted to SOME/IP serialization
[SWS_CM_80028]	-	restricted to signal-based serialization
[SWS_CM_80029]	[SWS_CM_10293]	identical
[SWS_CM_80030]	[SWS_CM_10379]	generalized
[SWS_CM_80031]	[SWS_CM_10296]	identical
[SWS_CM_80032]	[SWS_CM_10294]	restricted to SOME/IP serialization
[SWS_CM_80033]	-	restricted to signal-based serialization
[SWS_CM_80034]	[SWS_CM_10295]	identical
[SWS_CM_80035]	[SWS_CM_10297]	identical
[SWS_CM_80036]	[SWS_CM_10441]	identical
[SWS_CM_80037]	[SWS_CM_10298]	identical
[SWS_CM_80038]	[SWS_CM_10299]	identical
[SWS_CM_80039]	[SWS_CM_10300]	identical
[SWS_CM_80040]	[SWS_CM_10301]	identical
[SWS_CM_80041]	[SWS_CM_10302]	identical
[SWS_CM_80042]	[SWS_CM_10303]	identical
[SWS_CM_80043]	[SWS_CM_10304]	identical
[SWS_CM_80044]	[SWS_CM_10306]	identical
[SWS_CM_80045]	[SWS_CM_10307]	identical
[SWS_CM_80046]	[SWS_CM_10308]	identical
[SWS_CM_80047]	[SWS_CM_10309]	identical
[SWS_CM_80048]	[SWS_CM_10310]	identical
[SWS_CM_80049]	[SWS_CM_10311]	identical
[SWS_CM_80050]	[SWS_CM_10312]	identical
[SWS_CM_80051]	[SWS_CM_10428]	identical
[SWS_CM_80052]	[SWS_CM_10313]	identical
[SWS_CM_80053]	[SWS_CM_10314]	identical
[SWS_CM_80054]	[SWS_CM_10315]	identical
[SWS_CM_80055]	[SWS_CM_10357]	identical
[SWS_CM_80056]	[SWS_CM_10316]	identical
[SWS_CM_80057]	[SWS_CM_10442]	identical
[SWS_CM_80058]	[SWS_CM_10358]	identical
[SWS_CM_80059]	[SWS_CM_10429]	identical
[SWS_CM_80060]	[SWS_CM_10430]	identical
[SWS_CM_80061]	[SWS_CM_10317]	identical
[SWS_CM_80062]	[SWS_CM_10318]	identical
[SWS_CM_80063]	[SWS_CM_10319]	generalized
[SWS_CM_80064]	[SWS_CM_10320]	generalized
[SWS_CM_80065]	[SWS_CM_10321]	generalized



△

Signal-based binding	SOME/IP binding	Relation category
[SWS_CM_80066]	[SWS_CM_10322]	generalized
[SWS_CM_80067]	[SWS_CM_10323]	restricted to SOME/IP serialization
[SWS_CM_80068]	-	restricted to signal-based serialization
[SWS_CM_80069]	[SWS_CM_10324]	restricted to SOME/IP serialization
[SWS_CM_80070]	-	restricted to signal-based serialization
[SWS_CM_80071]	[SWS_CM_10325]	identical
[SWS_CM_80072]	[SWS_CM_10380]	generalized
[SWS_CM_80073]	[SWS_CM_10328]	identical
[SWS_CM_80074]	[SWS_CM_10326]	restricted to SOME/IP serialization
[SWS_CM_80075]	-	restricted to signal-based serialization
[SWS_CM_80076]	[SWS_CM_10327]	identical
[SWS_CM_80077]	[SWS_CM_10329]	identical
[SWS_CM_80078]	[SWS_CM_10443]	identical
[SWS_CM_80079]	[SWS_CM_10330]	identical
[SWS_CM_80080]	[SWS_CM_10331]	identical
[SWS_CM_80081]	[SWS_CM_10332]	identical
[SWS_CM_80082]	[SWS_CM_10333]	identical
[SWS_CM_80083]	[SWS_CM_10334]	identical
[SWS_CM_80084]	[SWS_CM_10335]	identical
[SWS_CM_80085]	[SWS_CM_10336]	identical
[SWS_CM_80086]	[SWS_CM_10338]	identical
[SWS_CM_80087]	[SWS_CM_10339]	identical
[SWS_CM_80088]	[SWS_CM_10340]	identical
[SWS_CM_80089]	[SWS_CM_10341]	identical
[SWS_CM_80090]	[SWS_CM_10342]	identical
[SWS_CM_80091]	[SWS_CM_10343]	identical
[SWS_CM_80092]	[SWS_CM_10344]	identical
[SWS_CM_80093]	[SWS_CM_10345]	identical
[SWS_CM_80094]	[SWS_CM_10346]	identical
[SWS_CM_80095]	[SWS_CM_10347]	identical
[SWS_CM_80096]	[SWS_CM_10348]	identical
[SWS_CM_80097]	[SWS_CM_10444]	identical
[SWS_CM_80098]	[SWS_CM_10349]	identical
[SWS_CM_80099]	[SWS_CM_10350]	identical
[SWS_CM_80100]	-	restricted to signal-based serialization
[SWS_CM_80101]	-	restricted to signal-based serialization
[SWS_CM_80102]	-	restricted to signal-based serialization
[SWS_CM_80103]	-	restricted to signal-based serialization

Table 7.4: Comparison Signal-based and SOME/IP network binding

One major difference between the SOME/IP network binding and the signal-based network binding is the serialization technology. While the SOME/IP network binding only supports SOME/IP serialized payload the signal-based 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).

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.

In figure 7.14 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.

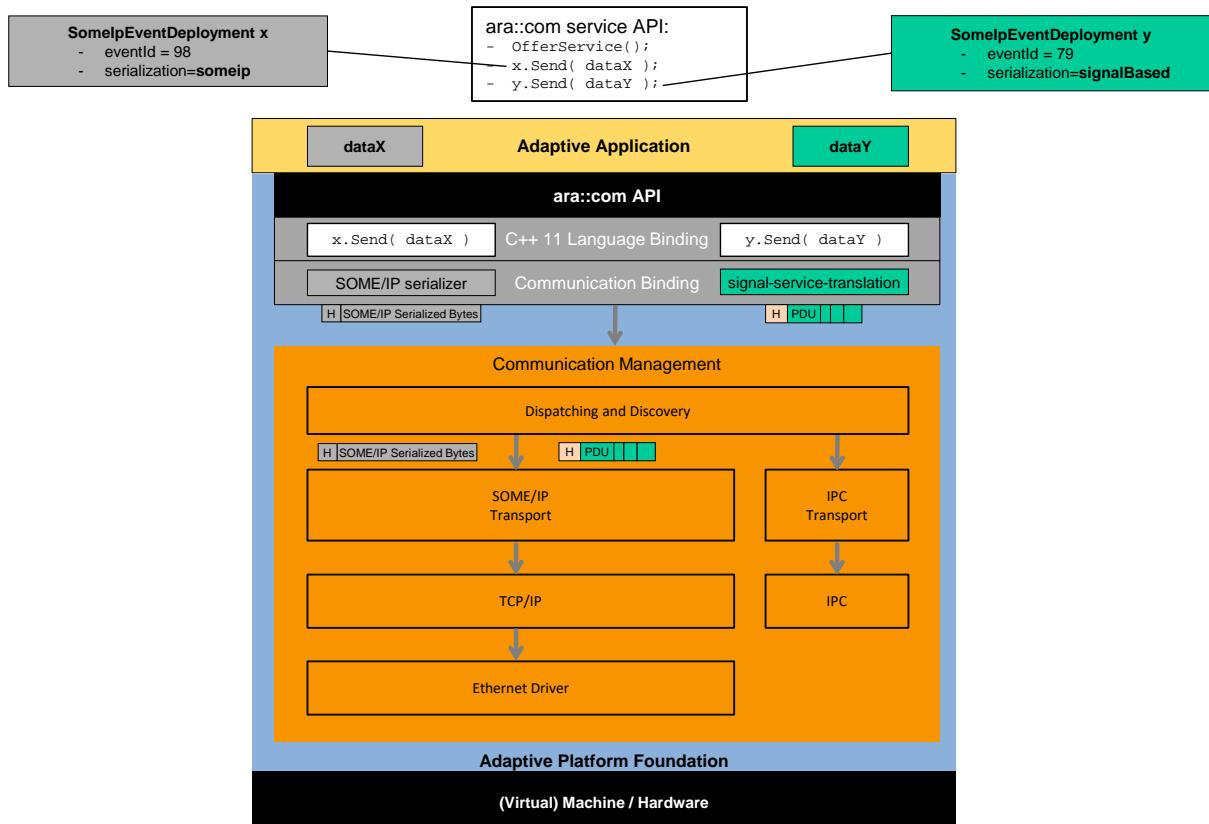


Figure 7.14: Example serialization settings

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

[SWS_CM_10174]{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).

[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 [4] (except for the serialization of signal-based payload).] (*RS_CM_00204, RS_CM_00205, RS_CM_00004*)

[SWS_CM_80002]{DRAFT} [All headers shall be encoded in network byte order Big Endian (MostSignificantByteFirst) [RFC 791].] (*RS_CM_00204, RS_SOMEIP_00026, RS_CM_00004*)

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

[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*)

7.5.2.1 Service Discovery

[SWS_CM_80005]{DRAFT} **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_CM_00004*)

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 [Initials-dDelayConfig](#) and [RequestResponseDelay](#). The sharing of timers is described in [TPS_MANI_03230].

[SWS_CM_80006]{DRAFT} 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 (INITIAL_DELAY_MIN, _MAX) followed by Repetition Wait phase (REPETITIONS_BASE_DELAY, REPETITIONS_MAX) and main phase (CYCLIC_OFFER_DELAY).]

(See also [PRS_SOMEIPSD_00395], [PRS_SOMEIPSD_00397], [PRS_SOMEIPSD_00399], [PRS_SOMEIPSD_00416], [PRS_SOMEIPSD_00435] and [PRS_SOMEIPSD_00752]) ([RS_CM_00204](#), [RS_CM_00102](#), [RS_SOMEIPSD_00024](#), [RS_CM_00004](#))

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 [Initials-dDelayConfig](#) and [RequestResponseDelay](#). The sharing of timers is described in [TPS_MANI_03231].

[SWS_CM_80007]{DRAFT} SOME/IP FindService message [The entries in the SOME/IP FindService message shall be as follows:

- The entry type shall be set to FindService (0x00).
- 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 "ANY" 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 "ANY" then 0xFFFF FFFF shall be used.

- TTL shall be derived from the Manifest where the `SomeipSdClientService-InstanceConfig` element that is referenced by the `RequiredSomeipServiceInstance` in the role `sdClientConfig` defines the `serviceFindTimeTo-Live`.
- Configuration Option shall be used in the find message if at least one `capabil-ityRecord` 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_-CM_00004*)

[SWS_CM_80008]{DRAFT} SOME/IP OfferService message [The entries in the SOME/IP OfferService message shall be as follows:

- The entry type shall be set to OfferService (0x01).
- 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 `Provided-SomeipServiceInstance` element defines the `serviceInstanceId` for the `SomeipServiceInterfaceDeployment` that is referenced by the `Provid-edSomeipServiceInstance` 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 `SomeipSdServerService-InstanceConfig` element that is referenced by the `ProvidedSomeipServiceInstance` in the role `sdServerConfig` defines the `serviceOfferTime-ToLive`.

- 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 configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.

- 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_CM_00004)

[SWS_CM_80009]{DRAFT} SOME/IP StopOffer message [The entries in the SOME/IP StopOffer message shall be as follows:

- The entry type shall be set to StopOfferService (0x01).
- 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_CM_00004](#))

[SWS_CM_80010]{DRAFT} 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_CM_00004](#))

[SWS_CM_80011]{DRAFT} 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](#))

[SWS_CM_80012]{DRAFT} 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](#) (0x06).
- 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 `eventId` 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 `Network-Endpoint` in the role `unicastNetworkEndpoint` where an IPv4 Address is configured in the `Ipv4Configuration` element.
- IPv6 Endpoint Option shall be sent if the offer message contains an IPv6 Endpoint Option. In this case the IPv6 Address sent in the IPv6 Endpoint Option of the `SubscribeEventgroup` message is configured in the Manifest where the `RequiredSomeipServiceInstance` element is mapped with the `ServiceInstanceToMachineMapping` to an `EthernetCommunicationConnector` of a Machine. The `EthernetCommunicationConnector` refers to a `Network-Endpoint` in the role `unicastNetworkEndpoint` where an IPv6 Address is configured in the `Ipv6Configuration` element.
- The Transport Layer Protocol used in the IPv4 Endpoint option and/or IPv6 Endpoint option shall be derived from the Manifest where the `SomeipEventGroup` points either to `SomeipEventDeployments` where the `transportProtocol` is set to `udp` or to `tcp`. The `SomeipServiceInstanceToMachineMapping` element that maps the `RequiredSomeipServiceInstance` to an `Ethernet-CommunicationConnector` 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 configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.
 - 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 configured

to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.

]([RS_CM_00204](#), [RS_CM_00200](#), [RS_CM_00103](#), [RS_SOMEIPSD_00006](#), [RS_CM_00004](#))

[SWS_CM_80013]{DRAFT} SOME/IP SubscribeEventgroupAck message [The entries in the SOME/IP SubscribeEventgroupAck message shall be as follows:

- The entry type shall be set to SubscribeEventgroupAck (0x07).
- Serviceld shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- Instanceld 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_CM_00004](#))

[SWS_CM_80014]{DRAFT} SOME/IP SubscribeEventgroupNack message [The entries in the SOME/IP SubscribeEventgroupNack message shall be as follows:

- The entry type shall be set to SubscribeEventgroupNack (0x07).
- Serviceld shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.

- InstancId 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_CM_00004](#))

[SWS_CM_80015]{DRAFT} 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 Events 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_CM_00004](#))

[SWS_CM_80016]{DRAFT} 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 (0x06).
- ServicId shall be set to the same value as in the SubscribeEventgroup message.
- InstancId 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_CM_00004](#))

7.5.2.2 Accumulation of messages

[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_80018]{DRAFT} 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_80019] and [SWS_CM_80020]) arrives and the data transmission starts.] *(RS_CM_00204)*

[SWS_CM_80019]{DRAFT} Configuration of a data accumulation on a `ProvidedServiceInstance` for transmission over UDP [For a `ProvidedServiceInstance` all `method` responses and `events` 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)*

[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 that 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_80018](#)], [[SWS_CM_80019](#)], and [[SWS_CM_80020](#)].

7.5.2.3 Execution context of message reception actions

In the following sections the term "upon reception" will be used to describe that fact that certain actions (e.g. the deserialization of the payload according to [[SWS_CM_80032](#)]) 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 separate execution context to leave room for potential optimizations of a concrete ara::com implementation.

The only restriction imposed here refers to the execution context of the `EventReceiveHandler` (see [[SWS_CM_00309](#)]). – Executing the `EventReceiveHandler` in the context of the `GetNewSamples` (see [[SWS_CM_00701](#)]) method is not allowed, since according to [[SWS_CM_00181](#)] the `EventReceiveHandler` shall use the `GetNewSamples` method to access the retrieved event data.

7.5.2.4 Handling Events

[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_80008](#)]) 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_80012]) has been exceeded.] (RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_CM_00004)

[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)

[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_80008]) as source address and source port for the transmission.] (RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042, RS_CM_00004)

[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_80013]) 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_80012]) 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.

[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 the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x000 (see [PRS_SOMEIP_00932] and [PRS_SOMEIP_00925]).

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

- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to NOTIFICATION (0x02).
- The Return Code (see [PRS_SOMEIP_00040]) is unused for event messages and thus (according to [PRS_SOMEIP_00040]) 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 serializer` the signal-based event handling applies.

As the PDUs containing the signal-based payload are interacting with the Classic platform without the SOME/IP transformation the header just contains the Message Id (i.e. ServiceID and Event ID).

[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 [5].

|(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.

[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.
- Verify that the Length (see [PRS_SOMEIP_00042]) is larger than 7.
- Use the Message Type (see [PRS_SOMEIP_00055]) which is set to NOTIFICATION (0x02) to determine that the received SOME/IP message is actually a SOME/IP event messages.
- 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` 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_00040]) 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 serializer` the signal-based event handling applies.

As the PDUs containing the signal-based payload are interacting with the Classic platform without the SOME/IP transformation the header just contains the Message Id (i.e. ServiceID and Event ID).

[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`.

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_80029]{DRAFT} 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](#), [RS_CM_00004](#)]

[SWS_CM_80030]{DRAFT} Silently discarding event messages for unsubscribed events [If the event identified according to [SWS_CM_80029] 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_80012]) has expired, then the received event message shall be silently discarded (i.e., [SWS_CM_80032], [SWS_CM_80033], [SWS_CM_80034], and [SWS_CM_80031] shall *not* be performed).] (RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_CM_00004)

[SWS_CM_80031]{DRAFT} 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_80029] this registered receive handler shall be invoked.] (RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_CM_00004)

[SWS_CM_80032]{DRAFT} Deserializing the SOME/IP serialized payload [If `SomeipEventDeployment.serializer` is set to `signalBased` then based on the event determined according to [SWS_CM_80029] 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)

[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_80029] 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 [5].] (RS_CM_00004)

[SWS_CM_80034]{DRAFT} 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_80029].] (RS_CM_00204, RS_CM_00202, RS_SOMEIP_00004, RS_CM_00004)

7.5.2.5 Handling Method Calls

[SWS_CM_80035]{DRAFT} 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_80008]) 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, RS_CM_00004)

[SWS_CM_80036]{DRAFT} 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, RS_CM_00004)

[SWS_CM_80037]{DRAFT} 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, RS_CM_00004)

[SWS_CM_80038]{DRAFT} 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 `SomeipServiceInstance-ToMachineMapping` as source address for the transmission. The `udpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_80037]) is UDP. The `tcpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_80037]) is TCP.] (RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00010, RS_CM_00004)

[SWS_CM_80039]{DRAFT} 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_80008]) 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_80037]) shall be used.] (RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_CM_00004)

[SWS_CM_80040]{DRAFT} 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 achieved by dynamically generating unique client IDs upon construction of the `ServiceProxy`.
- The Session ID (see [PRS_SOMEIP_00703]) shall be set to 0x0001 for the first call of a particular method by a given client and shall be incremented by 1 after each call performed by this client for the respective method (see [PRS_SOMEIP_00533]). Once the Session ID reaches 0xFFFF, it shall wrap around and start with 0x0001 again (see [PRS_SOMEIP_00521]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `REQUEST_NO_RETURN` (0x01) in case the `ClientServerOperation` referenced by `methodDeployment.method` contains a `fireAndForget` attribute which is set to `true`. The Message Type shall be set to `REQUEST` (0x00) otherwise.
- The Return Code (see [PRS_SOMEIP_00040]) 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, RS_CM_0004*)

[SWS_CM_80041]{DRAFT} 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 `SomeipMethodDeployments` of the `SomeipServiceInterfaceDeployment`.
- Verify that the Message Type (see [PRS_SOMEIP_00055]) is set to REQUEST_NO_RETURN (0x01) in case the the `ClientServerOperation` referenced by `methodDeployment.method` of the `SomeipMethodDeployment` with matching `methodId` attribute contains a `fireAndForget` attribute which is set to `true`. Verify that the Message Type is set to REQUEST (0x00) otherwise.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches `SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion`.
- Verify that the Return Code (see [PRS_SOMEIP_00040]) 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](#), [RS_CM_00004](#))

[[SWS_CM_80042](#)] {DRAFT} **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](#), [RS_CM_00004](#))

[[SWS_CM_80043](#)] {DRAFT} **Deserializing the payload** [Based on the method determined according to [[SWS_CM_80042](#)] 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](#), [RS_CM_00004](#))

[[SWS_CM_80044](#)] {DRAFT} **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_80042](#)] 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](#), [RS_CM_00004](#))

[SWS_CM_80045]{DRAFT} Invoke the method - polling [In case a Method-CallProcessingMode of kPoll has been passed to the constructor of the ServiceSkeleton (see [SWS_CM_00130]), the serialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke the service method (see [SWS_CM_00191]) identified according to [SWS_CM_80042] 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, RS_CM_00004*)

[SWS_CM_80046]{DRAFT} 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` of the service method (see [SWS_CM_80044] and [SWS_CM_80045]) 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, RS_CM_00004*)

[SWS_CM_80047]{DRAFT} 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, RS_CM_00004*)

[SWS_CM_80048]{DRAFT} 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 `udpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_80047]) is UDP. The `tcpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_80047]) is TCP.] (*RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00010, RS_CM_00004*)

[SWS_CM_80049]{DRAFT} 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_80038]) as destination address and destination port for the transmission.] (*RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_CM_00004*)

[SWS_CM_80050]{DRAFT} 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_80040](#)]).
- The Session ID (see [PRS_SOMEIP_00703]) shall be copied from the corresponding SOME/IP request message (see [[SWS_CM_80040](#)]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `ERROR` (0x81) in case the `ClientServerOperation` returned one of the possible `ApApplicationError`s referenced by the `ClientServerOperation` in role `possibleApError`³. The Message Type shall be set to `RESPONSE` (0x80) otherwise.
- The Return Code (see [PRS_SOMEIP_00040]) shall be set to `E_NOT_OK` (0x01) in case the `ClientServerOperation` raised one of the possible `ApApplicationError`s referenced by the `ClientServerOperation` in role `possibleApError`. 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 `ArgumentDataPrototype`s 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_80051](#)].

]([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](#), [RS_CM_00004](#))

³Note 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 `ApplicationError`s 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.

[SWS_CM_80051]{DRAFT} 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()`.

]([RS_CM_00004](#))

[SWS_CM_80052]{DRAFT} 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_80040](#)]).
- The Session ID (see [[PRS_SOMEIP_00703](#)]) matches the client from the corresponding SOME/IP request message (see [[SWS_CM_80040](#)]).

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](#), [RS_CM_00004](#))

[SWS_CM_80053]{DRAFT} 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`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](#), [RS_CM_00004](#))

[SWS_CM_80054]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80055]{DRAFT} Distinguishing errors from normal responses [The Message Type (see [PRS_SOMEIP_00055]) and the Return Code (see [PRS_SOMEIP_00040]) 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)⁴ 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](#), [RS_CM_00004](#))

[SWS_CM_80056]{DRAFT} Deserializing the payload - normal response messages [Based on the method determined according to [\[SWS_CM_80053\]](#) the Payload of the response message shall be deserialized according to the SOME/IP serialization rules. – Therefore the `ArgumentDataPrototypes`s with `direction` set to `inout` and `out` shall be serialized according to their order.] ([RS_CM_00204](#), [RS_CM_00212](#), [RS_CM_00213](#), [RS_SOMEIP_00007](#), [RS_SOMEIP_00028](#), [RS_CM_00004](#))

[SWS_CM_80057]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80058]{DRAFT} Identifying the right application error in a message with Message Type set to RESPONSE (0x80) [If the Return Code see [PRS_SOMEIP_00040]) 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 `ApApplicationError`s referenced in role `possibleApError` by the

⁴The additional case of SOME/IP response messages with a Return Code (see [PRS_SOMEIP_00040]) 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 `ApplicationErrors` are raised.

[ClientServerOperation](#) corresponding to the method determined according to [SWS_CM_80053], 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](#) by the [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].

If this computed [ApApplicationError.errorCode](#) attribute value does match more than one of the [ApApplicationError.errorCode](#) attributes of all [ApApplicationErrors](#) referenced in role [possibleApError](#) by the [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, RS_CM_00004)

Note: This is for backward compatibility to old servers using [RESPONSE](#) (0x80) even in case of application errors.

[SWS_CM_80059]{DRAFT} Identifying the right application error in a message with Message Type set to ERROR (0x81) [If the Return Code see [PRS_SOMEIP_00040]) contains a value equal to 0x01 (E_NOT_OK) then the corresponding [ApApplicationError](#) shall be identified by deserializing the Payload of the message according to the error payload format described in [SWS_CM_80051].] (RS_CM_00204, RS_SOMEIP_00008, RS_CM_00004)

[SWS_CM_80060]{DRAFT} Handling invalid messages with Message Type set to RESPONSE (0x81) [If the Return Code see [PRS_SOMEIP_00040]) 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_80051] or the application error identified by the deserialized [ApApplicationErrorDomain.value](#) and [ApApplicationError.errorCode](#) is not referenced in role [possibleApError](#) 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, RS_CM_00004)

[SWS_CM_80061]{DRAFT} 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_80055]) 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_80056] 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_80058] and [SWS_CM_80059] 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, RS_CM_00004)

[SWS_CM_80062]{DRAFT} 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, RS_CM_00004)

7.5.2.6 Handling Fields

[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_80008]) 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_80012]) 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)

[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)

[SWS_CM_80065]{DRAFT} Source of an event message [The source address and the source port of the event message shall be set according to [SWS_CM_80023].]

(*RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042, RS_CM_00004*)

[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.

[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 [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x000 (see [PRS_SOMEIP_00932] and [PRS_SOMEIP_00925]).

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

- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to NOTIFICATION (0x02).
- The Return Code (see [PRS_SOMEIP_00040]) is unused for event messages and thus (according to [PRS_SOMEIP_00040]) 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 serializer** the signal-based event handling applies.

As the PDUs containing the signal-based payload are interacting with the Classic platform without the SOME/IP transformation the header just contains the Message Id (i.e. ServiceID and Event ID).

[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 [5].

](*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.

[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 serializer` the signal-based serialized event handling applies.

As the PDUs containing the signal-based payload are interacting with the Classic platform without the SOME/IP transformation the header just contains the Message Id (i.e. ServiceID and Event ID).

[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_80071]{DRAFT} 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.notifiers` 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](#), [RS_CM_00004](#))

[SWS_CM_80072]{DRAFT} Silently discarding event messages for unsubscribed events [If the event identified according to [SWS_CM_80071] 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_80012]) has expired, then the received event message shall be silently discarded (i.e., [SWS_CM_80074], [SWS_CM_80101], [SWS_CM_80076], and [SWS_CM_80073] shall not be performed).] ([RS_CM_00204](#), [RS_CM_00203](#), [RS_SOMEIP_00004](#), [RS_SOMEIP_00009](#), [RS_CM_00004](#))

[SWS_CM_80073]{DRAFT} 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_80071] this registered receive handler shall be invoked.] ([RS_CM_00204](#), [RS_CM_00203](#), [RS_SOMEIP_00004](#), [RS_SOMEIP_00009](#), [RS_CM_00004](#))

[SWS_CM_80074]{DRAFT} Deserializing the SOME/IP serialized payload [If `SomeipEventDeployment.serializer` is set to `signalBased` then based on the event determined according to [SWS_CM_80071] 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](#))

[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_80071] 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 [5].] ([RS_CM_00004](#))

[SWS_CM_80076]{DRAFT} Providing the received event data [The serialized 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_80071].] ([RS_CM_00204](#), [RS_CM_00202](#), [RS_SOMEIP_00004](#), [RS_SOMEIP_00009](#), [RS_CM_00004](#))

[SWS_CM_80077]{DRAFT} 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_80008]) 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](#), [RS_CM_00004](#))

[SWS_CM_80078]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80079]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80080]{DRAFT} 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_80038].] ([RS_CM_00204](#), [RS_CM_00212](#), [RS_CM_00213](#), [RS_SOMEIP_00007](#), [RS_SOMEIP_00009](#), [RS_SOMEIP_00010](#), [RS_CM_00004](#))

[SWS_CM_80081]{DRAFT} 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_80039].] (RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_CM_00004)

[SWS_CM_80082]{DRAFT} 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_00040]) 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, RS_CM_00004)

[SWS_CM_80083]{DRAFT} 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_00040]) 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](#), [RS_CM_00004](#))

[SWS_CM_80084]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80085]{DRAFT} Deserializing the payload [Based on the method determined according to [\[SWS_CM_80084\]](#) 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](#), [RS_CM_00004](#))

[SWS_CM_80086]{DRAFT} 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 serialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke a registered SetHandler resp. GetHandler (see [SWS_CM_00114] and [SWS_CM_00116]) of the Field class as a consequence to the reception of the SOME/IP request message.] (RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_CM_00004)

[SWS_CM_80087]{DRAFT} 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 serialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke a registered SetHandler resp. GetHandler (see [SWS_CM_00114] and [SWS_CM_00116]) of the Field class 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, RS_CM_00004)

[SWS_CM_80088]{DRAFT} 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, RS_CM_00004)

[SWS_CM_80089]{DRAFT} 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, RS_CM_00004)

[SWS_CM_80090]{DRAFT} 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_80048].] (RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010, RS_CM_00004)

[SWS_CM_80091]{DRAFT} 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_80049].] (RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_CM_00004)

[SWS_CM_80092]{DRAFT} 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_80040]).
- The Session ID (see [PRS_SOMEIP_00703]) shall be copied from the corresponding SOME/IP request message (see [SWS_CM_80040]).
- 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_00040]) 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](#), [RS_CM_00004](#))

[SWS_CM_80093]{DRAFT} Checks for a received SOME/IP response message Upon reception of a SOME/IP response message the checks defined in [SWS_CM_80052] 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](#), [RS_CM_00004](#))

[SWS_CM_80094]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80095]{DRAFT} Discarding orphaned responses [Orphaned responses shall be discarded according to [SWS_CM_80054].] ([RS_CM_00204](#), [RS_CM_00212](#), [RS_CM_00213](#), [RS_CM_00004](#))

[SWS_CM_80096]{DRAFT} Deserializing the payload [Based on the method determined according to [SWS_CM_80094] 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](#), [RS_CM_00004](#))

[SWS_CM_80097]{DRAFT} 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](#), [RS_CM_00004](#))

[SWS_CM_80098]{DRAFT} 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 serialized 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](#), [RS_CM_00004](#))

[SWS_CM_80099]{DRAFT} Invoke the notification function [Any registered notification function shall be invoked according to [SWS_CM_80062].] ([RS_CM_00204](#), [RS_CM_00212](#), [RS_CM_00213](#), [RS_CM_00215](#), [RS_SOMEIP_00007](#), [RS_SOMEIP_00009](#), [RS_CM_00004](#))

7.5.2.7 Serialization of Payload

The serialization technology is defined by the attribute `SomeipEventDeployment.serializer`. If the attribute is set to `signalBased` then the signal-service-translation is responsible for the handling of the serialization. If the attribute is set to `someip` then the SOME/IP serializer (see section 7.5.1.7) 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 [7.5.1.7](#).

[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 [5].] ([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](#))

[SWS_CM_80103]{DRAFT} Init value for field elements [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 discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).] ([RS_CM_00004](#))

7.5.3 DDS Network binding

[SWS_CM_11000] [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](#))

7.5.3.1 Service Discovery

[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`.
- [\[SWS_CM_11029\]](#) 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 `ClientServerOperations` 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 **Fields** defined in the **ServiceInterface** in the role **field** with **hasGetter** and/or **hasSetter** attributes set to true via getter/setter invocation.
- [SWS_CM_11004] 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 above⁵. If the search is successful, the binding shall assign the DomainParticipant found to the Service⁶; 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**.

⁵The 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.

⁶The rules specified in this binding ensure the creation of only one DomainParticipant for a given Domain and set of QoS settings (**qosProfile**).

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 [\[SWS_CM_11015\]](#).

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_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 [7.5.3.3](#).

[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 `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 [\[SWS_CM_11002\]](#). 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 `fields` in the `ServiceInterface` with `hasGetter` and/or `hasSetter` set to `true`.

In practice, this implies assigning a DDS Request Topic and a DDS Reply Topic to the DDS Service according to the mapping rules specified in [\[SWS_CM_11144\]](#). Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create new DDS Request and Reply Topics to represent the DDS Service as specified in [\[SWS_CM_11144\]](#).

Once the corresponding DDS Request and Reply Topics are ready for use, the DomainParticipant assigned to the Service Instance shall create:

- [\[SWS_CM_11149\]](#) A DDS DataReader of the DDS Request Topic to handle requests using the DDS Subscriber created in [\[SWS_CM_11002\]](#).
- [\[SWS_CM_11150\]](#) 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 [7.5.3.4](#).

[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_11004] Adding Service and Service Instance IDs to the DDS Domain-Participant'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 the Service and Instance IDs 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 and Service Instance IDs shall be encoded in the USER_DATA QoS Policy in string format according to the following pattern:

```
"ara.com://services/<svcId>_<svcInId> [&<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`.

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

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

]([RS_CM_00204](#), [RS_CM_00200](#), [RS_CM_00101](#))

[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 `events` in the `ServiceInterface` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters and DataReaders associated with the `ClientServerOperations` defined in the role `method` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters associated with `fields` in the `ServiceInterface` with their `hasNotifier` attribute set to `true` created in previous calls to the `OfferService()` method.

- It shall remove all DDS DataWriters and DataReaders associated with the `fields` in the `ServiceInterface` with `hasGetter` and/or `hasSetter` attributes set to `true` created in previous calls to the `OfferService()` method.

]([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 "ANY", 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 "ANY", 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 criteria⁷. 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:
 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

⁷These 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.

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`⁸.

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`⁹.

]([RS_CM_00204](#), [RS_CM_00200](#))

⁸Note 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.

⁹Note 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.

[SWS_CM_11014] Unbinding a BuiltinParticipantListener from a DDS Domain-Participant [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](#))

7.5.3.2 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, where the `DdsEventDeployment` element defines the `topicName`.
- The Topic Data Type shall be defined as specified in [\[SWS_CM_11016\]](#), and shall be registered under the equivalent data type's 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 definition¹⁰:

```
1 struct <eventTypeName>EventType {  
2     @key uint16 instance_id;  
3     @external <eventTypeName> data;  
4 };
```

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 a reference (per language mapping of the `@external` annotation) to the actual value of the `event`, which shall be constructed and encoded according to the DDS serialization rules.

] ([RS_CM_00204, RS_CM_00201](#))

The DDS serialization rules are defined in section [7.5.3.5](#).

¹⁰DDS 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).

[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 [7.5.3.5](#).

[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 [\[SWS_CM_11015\]](#)). 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. 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<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](#))

7.5.3.3 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 `ClientServerOperations`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 `methods` 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 `ClientServerOperations` 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, where the `DdsRpcServiceDeployment` element associated with the methods defines the `requestTopicName`.
- 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, where the `DdsRpcServiceDeployment` element associated with the methods defines the `replyTopicName`.
- 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 metadata a Call Structure with data. The IDL definition of the Request Topic data type is the following:

```
1 struct <svcId>Method_Request {
2     dds::rpc::RequestHeader header;
3     <svcId>Method_Call data;
4 };
```

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:

```

1 union <svcid>Method_Call switch(int32) {
2     default:
3         dds::rpc::UnknownOperation unknownOp;
4     case <svcid>Method_<method0Name>_Hash:
5         <svcid>Method_<method0Name>_In <method0Name>;
6     case <svcid>Method_<method1Name>_Hash:
7         <svcid>Method_<method1Name>_In <method1Name>;
8     // ...
9     case <svcid>Method_<methodNName>_Hash:
10        <svcid>Method_<methodNName>_In <methodNName>;
11 }

```

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:

```

1 struct <svcid>Method_<methodName>_In {
2     <ArgumentDataPrototype[0]>;
3     <ArgumentDataPrototype[1]>;
4     // ...
5     <ArgumentDataPrototype[n]>;
6 }

```

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 metadata and a Return Structure with data. The IDL definition of the Reply Topic data type is the following:

```
1 struct <svcId>Method_Reply {  
2     dds::rpc::ReplyHeader header;  
3     <svcId>Method_Return data;  
4 };
```

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 <sv-`
`cId>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:

```
1 union <svcId>Method_Return switch(int32) {  
2     default:  
3         dds::rpc::UnknownOperation unknownOp;  
4     case <svcId>Method_<methodName>_Hash:  
5         <svcId>Method_<methodName>_Result <methodName>;  
6     case <svcId>Method_<methodName>_Hash:  
7         <svcId>Method_<methodName>_Result <methodName>;  
8     // ...  
9     case <svcId>Method_<methodName>_Hash:  
10        <svcId>Method_<methodName>_Result <methodName>;  
11 }
```

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:

```
1 union <svcId>Method_<methodName>_Result switch(int32) {  
2     case dds::RETCODE_OK:  
3         <svcId>Method_<methodName>_Out result;  
4     case dds::RETCODE_ERROR:  
5         ara::core::ErrorCode error;
```

```
6  };
```

Lastly, as defined in section 7.5.1.1.5 of [21], the <sv-
cId>Method_<methodName>_Out structure be constructed as follows:

- The structure shall contain as members all the `ArgumentDataPrototypes` 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.
- For non-void methods, the structure shall include a last member named `return_` of the method's return type. If the method has an argument named `return_`, the member shall be renamed according to the rules specified in section 7.5.1.1.5 of [21]. If the return type of the method is of `ara::core::Result<ValueType, ErrorType>` then the `ValueType` is considered as `<ReturnType>`.
- If the method has no return value, 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 <svId>Method_<methodName>_Out is the following:

```
1 struct <svId>Method_<methodName>_Out {  
2     <ArgumentDataPrototype[0]>;  
3     <ArgumentDataPrototype[1]>;  
4     // ...  
5     <ArgumentDataPrototype[n]>;  
6     [<ReturnType> return_];  
7 };
```

The resulting Reply Topic data type shall be encoded according to the DDS serialization rules. Unions, such as the <svId>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]:

```
1 module ara { module core {  
2  
3     struct ErrorCode {  
4         uint64 error_domain_value;  
5         int32 error_code;  
6     };  
7  
8 };}; // module ara::core
```

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.] ([RS_CM_00204](#))

The DDS serialization rules are defined in section [7.5.3.5](#).

[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.

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 `DdsMethodQosProps` 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.

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 `DdsMethodQosProps` 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](#)]).

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 `DdsMethodQosProps` 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]).]

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 `DdsMethodQosProps` 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 [7.5.3.5](#).

[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. 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_00203](#), [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

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. 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.

[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 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 <sv-cId>Method_Call and provide the destination method with the specific ArgumentDataPrototypes 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 <sv-cId>Method_Call union and provide the destination method with the specific ArgumentDataPrototypes 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 ApApplicationErrors 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 [in-out](#) 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_200](#), [RS_CM_00212](#), [RS_CM_00213](#))

The DDS serialization rules are defined in section 7.5.3.5.

7.5.3.4 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, where the [DdsEventDeployment](#) element defined in the [DdsFieldDeployment](#) in the role [notifier](#) defines the [topicName](#).
- 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:

```
1 struct <fieldName>FieldNotifierType {
2     @key uint16 instance_id;
3     @external <fieldName> data;
4 };
```

Where:

<**fieldName**> is the Cpp Implementation Data Type symbol (see section 8.1.2.5.2).

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 a reference (per language mapping of the @external annotation) to the actual value of the **field**, which shall be constructed and encoded according to the DDS serialization rules.

]([RS_CM_00204](#), [RS_CM_00201](#))

The DDS serialization rules are defined in section 7.5.3.5.

[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 7.5.3.5.

[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_11009](#)]. 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](#)]). 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 `DdsField-QosProps` 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, where the *DdsRpcServiceDeployment* element in the role *ddsRpcService* of the field's get and *set* methods defines the *requestTopicName*.
- 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, where the *DdsRpcServiceDeployment* element in the role *ddsRpcService* of the field's get and *set* methods defines the *replyTopicName*.
- 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 metadata 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:

```
1 struct <svcId>Field_Request {
2     dds::rpc::RequestHeader header;
3     <svcId>Field_Call data;
4 };
```

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 `Fields` 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:

```

1 union <svcId>Field_Call switch(int32) {
2     default:
3         dds::rpc::UnknownOperation unknownOp;
4     case <svcId>Field_Get<Field0Name>_Hash:
5         <svcId>Field_Get<Field0Name>_In get<Field0Name>;
6     case <svcId>Field_Set<Field0Name>_Hash:
7         <svcId>Field_Set<Field0Name>_In set<Field0Name>;
8     case <svcId>Field_Get<Field1Name>_Hash:
9         <svcId>Field_Get<Field1Name>_In get<Field1Name>;
10    case <svcId>Field_Set<Field1Name>_Hash:
11        <svcId>Field_Set<Field1Name>_In set<Field1Name>;
12    // ...
13    case <svcId>Field_Get<FieldNName>_Hash:
14        <svcId>Field_Get<FieldNName>_In get<FieldNName>;
15    case <svcId>Field_Set<FieldNName>_Hash:
16        <svcId>Field_Set<FieldNName>_In set<FieldNName>;
17 }

```

According to 7.5.1.1.4 of [21], `<svcId>Field_Set<FieldName>_In` structures shall contain as member, the corresponding `StdCppImplementation-DataType` 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 metadata 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:

```

1 struct <svcId>Field_Reply {
2     dds::rpc::ReplyHeader header;
3     <svcId>Field_Return data;
4 }

```

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 `Fields` 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 `<sv-
cId>Field_Get<FieldName>_Result` for getter methods and `<sv-
cId>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:

```
1 union <svcId>Field_Return switch(int32) {  
2     default:  
3         dds::rpc::UnknownOperation unknownOp;  
4     case <svcId>Field_Get<Field0Name>_Hash:  
5         <sv-  
cId>Field_Get<Field0Name>_Result get<Field0Name>;  
6     case <svcId>Field_Set<Field0Name>_Hash:  
7         <sv-  
cId>Field_Set<Field0Name>_Result set<Field0Name>;  
8     case <svcId>Field_Get<Field1Name>_Hash:  
9         <sv-  
cId>Field_Get<Field1Name>_Result get<Field1Name>;  
10    case <svcId>Field_Set<Field1Name>_Hash:  
11        <sv-  
cId>Field_Set<Field1Name>_Result set<Field1Name>;  
12    // ...  
13    case <svcId>Field_Get<FieldNName>_Hash:  
14        <sv-  
cId>Field_Get<FieldNName>_Result get<FieldNName>;  
15    case <svcId>Field_Set<FieldNName>_Hash:
```

```

16     <svcId>Field_Set<FieldName>_Result set<FieldName>;
17 }

```

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:

```

1 union <svcId>Field_Get<FieldName>_Result switch(int32) {
2 case dds::RETCODE_OK:
3     <svcId>Field_Get<FieldName>_Out result_;
4 }

```

Likewise, the IDL representation of `<svcId>Field_Set<FieldName>_Result` is the following:

```

1 union <svcId>Field_Set<FieldName>_Result switch(int32) {
2 case dds::RETCODE_OK:
3     <svcId>Field_Set<FieldName>_Out result_;
4 }

```

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 `fields` of the `ServiceInterface` (see [SWS_CM_11145]) upon proxy instantiation.

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 `DdsField-QosProps` 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.]

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 `DdsField-QosProps` 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 [\[SWS_CM_11145\]](#)).]

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 `DdsField-QosProps` 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 [fields](#) of the [ServiceInterface](#) (see [[SWS_CM_11146](#)]).]

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 [DdsField-QosProps](#) 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 [7.5.3.5](#).

[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.relatedRequestId` of the received sample with the original `requestId` that was sent in [SWS_CM_11151]¹². 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].]

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

¹²See footnote 11.

[[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 operation¹³ as a result of a Get() operation where no GetHandler was previously registered.

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:

¹³An internal lookup operation to retrieve the current value of a field.

- 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, and 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 7.5.3.5.

7.5.3.5 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)

7.5.3.5.1 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 7.5 provides the equivalent DDS PRIMITIVE_TYPES for the primitive `StdCppImplementationDataType`s with category VALUE defined in [13].] (RS_CM_00204, RS_CM_00200, RS_CM_00102)

Type	DDS Type	Remark
<code>boolean</code>	<code>Boolean</code>	
<code>uint8_t</code>	<code>Byte</code>	Shall be encoded as a Byte type (opaque 8-bit type).
<code>uint16_t</code>	<code>UInt16</code>	
<code>uint32_t</code>	<code>UInt32</code>	
<code>uint64_t</code>	<code>UInt64</code>	
<code>int8_t</code>	<code>Byte</code>	Shall be encoded as a Byte type (opaque 8-bit type).
<code>int16_t</code>	<code>Int16</code>	
<code>int32_t</code>	<code>Int32</code>	
<code>int64_t</code>	<code>Int64</code>	
<code>float</code>	<code>Float32</code>	
<code>double</code>	<code>Float64</code>	

Table 7.5: `StdCppImplementationDataType`s with category VALUE supported for serialization

7.5.3.5.2 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_CM_00424]) in bits.] ([RS_CM_00204](#), [RS_CM_00201](#), [RS_CM_00202](#), [RS_CM_00211](#))

7.5.3.5.3 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](#))

7.5.3.5.4 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](#))

7.5.3.5.5 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 `CppType`s 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 `CppType` of category ARRAY [A `CppType` 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](#))

7.5.3.5.6 Associative Maps

[SWS_CM_11049] DDS serialization of `CppType` of category ASSOCIATIVE_MAP [`CppType` 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](#))

7.5.3.5.7 Variant

[SWS_CM_11050] DDS serialization of `CppType` of category VARIANT [`CppType` 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](#))

7.6 Security

In the following chapter the behavior according to the meta model of access control and secure communication shall be described.

7.6.1 Access Control

The following assumptions have to be held true to realize access control:

1. Communication between two applications must be realized by using `ara::com` interfaces Communication Management to enable access control.
2. Process separation as defined in [\[SWS_CM_90004\]](#)

[SWS_CM_90004]{DRAFT} Process separation of network and language binding for access control [The application with the language binding part of proxies and the

network binding part of proxies shall be located in different processes.] ([RS_SEC_03003](#), [RS_SEC_03005](#), [RS_SEC_05019](#))

[SWS_CM_90001]{DRAFT} Restrictions on executing methods [The invocation of a method by an application shall be executed depending the existence of `ComMethodGrant`, `ComFieldGrant` with the role attribute set to `FieldAccessEnum.getter` or `FieldAccessEnum.setter`. From a temporal perspective the enforcement of the capability 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](#)])

A failure of the capability enforcement (i.e., an invocation without corresponding capability modeling) shall be handled according to [SWS_CORE_00002].] ([RS_SEC_03002](#), [RS_SEC_03008](#), [RS_SEC_03010](#))

[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` with the role attribute set to `FieldAccessEnum.setter`. From a temporal perspective the enforcement of the capability 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 capability enforcement (i.e., the triggering of an event without appropriate capability modeling) shall cause the event to be dropped silently.] ([RS_SEC_03002](#), [RS_SEC_03008](#), [RS_SEC_03010](#))

[SWS_CM_90003]{DRAFT} Restrictions on receiving events [Subscribing to event notifications shall be enabled depending on the existence of `ComEventGrant` or `ComFieldGrant` with the role attribute set to `FieldAccessEnum.getter`. From a temporal perspective the enforcement of the capability 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 capability enforcement (i.e., the subscription to an event without appropriate capability modeling) shall cause the subscription to the event to be dropped silently.] ([RS_SEC_03002](#), [RS_SEC_03008](#), [RS_SEC_03010](#))

[SWS_CM_90005]{DRAFT} Restrictions on offering services [Offering a service instance shall be enabled depending on the presence of a `ComOfferServiceGrant`.

From a temporal perspective the enforcement of the capability shall take place after the invocation of the following method:

- the constructor of the respective ServiceSkeleton class (see [SWS_CM_00130])

A failure of the capability enforcement (i.e., an invocation without corresponding modeling) shall be handled according to [SWS_CORE_00002].] (RS_SEC_03002, RS_SEC_03008, RS_SEC_03010)

[SWS_CM_90006]{DRAFT} Restrictions on using services [Using a service instance shall be enabled depending on the presence of a ComFindServiceGrant. From a temporal perspective the enforcement of the capability shall take place after the invocation of the following method:

- the constructor of the respective ServiceProxy class (see [SWS_CM_00131])

A failure of the capability enforcement (i.e., an invocation without corresponding modeling) shall be handled according to [SWS_CORE_00002].] (RS_SEC_03002, RS_SEC_03008, RS_SEC_03010)

[SWS_CM_90007]{DRAFT} 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 capability shall take place after the invocation of the following method:

- the Connect() method of the respective RawDataStream class (see [SWS_CM_10468])

] (RS_SEC_03002, RS_SEC_03008, RS_SEC_03010)

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.

7.6.2 Secure Communication

7.6.2.1 SOME/IP Network binding

SOME/IP communication can be transported via TCP and UDP. Therefore different security mechanism have to be available to secure the SOME/IP communication. The following security protocols are currently supported:

- TLS 1.2 (see [RFC5246])

- DTLS 1.2 (see [RFC6347])
- SecOC
- IPSec

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])

SOME/IP supports one-to-many (unicast) and many-to-many (multicast) communication paradigms. These paradigms may switch at runtime for events (see [multicast-Threshold](#)).

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.

[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 `AdaptivePlatformServiceInstances` 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 input. 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`.

In addition it is possible to define which elements of the `ServiceInterface` of the particular `AdaptivePlatformServiceInstance` needs to go via the secured channel. The selection of `ServiceInterface` elements is done by the `ServiceInterfaceElementSecureComConfig` that is aggregated by `AdaptivePlatformServiceInstance`.

The following configuration in the `ServiceInterfaceElementSecureComConfig` is applicable:

- **Methods**

The roles `methodCall` and `methodReturn` identify the `method(s)` that shall be sent using the referenced secure channel.

- **Events**

The role `event` identifies the `event(s)` that shall be sent using the referenced secure channel.

- **Fields**

The roles `fieldNotifier`, `getterCall`, `getterReturn`, `setterCall` and `setterReturn` identify the `event and method(s)` that shall be sent using the referenced secure channel.

]([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.

A (D)TLS secure channel may provide authenticity, integrity and confidentiality.

[SWS_CM_90103]{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 `secureComProps-ForTcp` 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 “tcp” by `transportProtocol` in the associated `SomeipMethodDeployment`.

]([RS_SEC_04001](#))

[SWS_CM_90104]{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 `secureComProps-ForUdp` by a `ServiceInstanceToMachineMapping` and a `method` of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this `method` is configured for transmission over “udp” by `transportProtocol` in the associated `SomeipMethodDeployment`.

]([RS_SEC_04001](#))

[SWS_CM_90105]{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 `secureComProps-ForTcp` by a `ServiceInstanceToMachineMapping` and an `event` of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and

this event is configured for transmission over “tcp” by [transportProtocol](#) in the associated [SomeipEventDeployment](#).

]([RS_SEC_04001](#))

[SWS_CM_90106]{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 [SomeipEventDeployment](#).

]([RS_SEC_04001](#))

[SWS_CM_90107]{DRAFT} TLS secure channel for fields [The requirements [\[SWS_CM_90103\]](#), [\[SWS_CM_90104\]](#), [\[SWS_CM_90105\]](#) and [\[SWS_CM_90106\]](#) apply to fields in the same manner, since fields are a composition of methods and events.]([RS_SEC_04001](#))

[SWS_CM_90120]{DRAFT} TLS client role of a Proxy [The TLS secure channel shall be associated with the respective [Proxy](#) and the implementation shall act as a TLS client, if the [AdaptivePlatformServiceInstance](#) referenced in

- [\[SWS_CM_90103\]](#)
- [\[SWS_CM_90104\]](#)
- [\[SWS_CM_90105\]](#)
- [\[SWS_CM_90106\]](#)
- [\[SWS_CM_90107\]](#)

is a [RequiredApServiceInstance](#).]([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

- [\[SWS_CM_90103\]](#)
- [\[SWS_CM_90104\]](#)
- [\[SWS_CM_90105\]](#)
- [\[SWS_CM_90106\]](#)
- [\[SWS_CM_90107\]](#)

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 [Proxies](#) 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 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, error handling according to [\[SWS_CORE_00001\]](#) shall be done as if the peer was unreachable.] ([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 `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 `Update()` method of the respective `Field` class (see [SWS_CM_00119])

] (RS_SEC_04004)

A SecOC secure channel may provide authenticity and integrity.

[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 `Someip-MethodDeployment`.

] (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 "tcp" by `transportProtocol` in the associated `Someip-MethodDeployment`.

] (RS_SEC_04001)

[SWS_CM_90109]{DRAFT} SecOC secure channel for events 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 of the `AdaptivePlatformServiceInstance` is selected for transmission

over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this event 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 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 of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this event 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](#))

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*)

7.6.2.2 DDS

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 PSMs¹⁴, and Shared Memory PSMs for Inter-Process Communication (IPC).

For consistency with the secure channel modeling and secure communication mechanisms specified in 7.6.2.1, 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.

Implementers of the DDS Network Binding who may want to provide transport-independent secure communication and fine-grained access control at the DDS Domain- and Topic-level may use the mechanisms defined in the DDS Security specification [24] in accordance with [*SWS_CM_90210*].

[SWS_CM_90201]{DRAFT} Secure channel creation [Secure channels shall be created as specified in [*SWS_CM_90101*].] (*RS_SEC_04001*)

[SWS_CM_90202]{DRAFT} Using secure channels [Secure channels shall be used as specified in [*SWS_CM_90102*].] (*RS_SEC_04001, RS_SEC_04003*)

[SWS_CM_90203]{DRAFT} TLS secure channel for methods using reliable transport [A TLS secure channel shall be created and used if:

¹⁴A standard TCP PSM for DDS-RTPS is under development, the RFP document is publicly available at the Object Management Group website: <https://www.omg.org/cgi-bin/doc.cgi?mars/2017-9-24>.

- a `TlsSecureComProps` instance is referenced in the role `secureComProps-ForTcp` by a `ServiceInstanceToMachineMapping` and 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 `DdsMethodDeployment`.

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 `secureComProps-ForUdp` by a `ServiceInstanceToMachineMapping` and a method of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this method is configured for transmission over “udp” by `transportProtocol` in the associated `DdsMethodDeployment`.

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 `secureComProps-ForTcp` by a `ServiceInstanceToMachineMapping` and an event of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this event is configured for transmission over “tcp” by `transportProtocol` in the associated `DdsEventDeployment`.

The DataReaders and DataWriters associated with the event shall be configured to operate over TLS.] ([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)

[SWS_CM_90210]{DRAFT} Using the DDS Security standard plug-ins in the Adaptive Platform [Implementers of the DDS binding may use the standard DDS Security plug-ins specified in [24] instead of the security mechanisms defined in this document. The DDS Security plug-ins enable transport-independent secure communication and fine-grained access control on the DDS Domains and Topics that are created as a result of the DDS network binding. These mechanisms shall be configured using the standard Governance and Permission files specified in [24].]

When using DDS Security instead of the mechanisms specified in this document, `DdsProvidedServiceInstances` and `DdsRequiredServiceInstances` shall contain no `secureComConfig` properties to ensure that the secure communication relies solely on DDS Security mechanisms.] (RS_SEC_04001)

7.6.2.3 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_90211]{DRAFT} 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]{DRAFT} 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)

[SWS_CM_90213]{DRAFT} 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 `tcpPort` in the `EthernetRawDataStreamMapping`

](*RS_SEC_04001*)

[SWS_CM_90214]{DRAFT} 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 `udpPort` or `multicastUdpPort` in the `EthernetRawDataStreamMapping`

](*RS_SEC_04001*)

[SWS_CM_90215]{DRAFT} 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*)

7.7 Communication API

In the following chapter the functional API specification shall be described.

7.7.1 Offer service

For the service offering C++ API reference, see chapter [8.1.3.2](#).

[SWS_CM_00102]{DRAFT} Uniqueness of offered service [The Communication Management shall check the offered service for uniqueness. If the implementation detects a duplication (i.e., a service with the same `ServiceIdentifier` and `InstanceIdentifier` is already registered), it shall perform error handling according to [\[SWS_CORE_00001\]](#).](*RS_CM_00200, RS_CM_00101*)

[SWS_CM_00103]{DRAFT} 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.](*RS_CM_00101*)

7.7.2 Service skeleton creation

For the service skeleton creation C++ API reference, see chapter [8.1.3.3](#).

[SWS_CM_10410]{DRAFT} InstanceIdentifier check during the creation of service skeleton [The Communication Management shall check the value of the InstanceIdentifier argument: the identifier shall be unique, using the same instance identifier for the creation of more than one skeleton instance of the same service shall cause error handling according to [SWS_CORE_00001].] ([RS_CM_00101](#))

[SWS_CM_10450]{DRAFT} 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 according to [SWS_CORE_00001].] ([RS_CM_00101](#), [RS_AP_00137](#))

[SWS_CM_10451]{DRAFT} 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.

Failing checks and using the same InstanceIdentifier for the creation of more than one skeleton instance of the same service shall be handled according to [SWS_CORE_00001].] ([RS_CM_00101](#))

7.7.3 Processing of service methods

For the processing of service methods C++ API reference, see chapter [8.1.3.6](#).

[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](#))

7.7.4 Registering get handlers for fields

For the registering get handlers for fields C++ API reference, see chapter [8.1.3.7](#).

[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](#))

7.7.5 Registering set handlers for fields

For the registering set handlers for fields C++ API reference, see chapter [8.1.3.8](#).

[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 must 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 [\[SWS_CM_00101\]](#)), error handling according to [SWS_CORE_00001] shall be performed, 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.

]([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, error handling according to [SWS_CORE_00001] shall be performed, if for at least one contained `Field` having `hasSetter = true` no `SetHandler` (see [[SWS_CM_00116](#)]) has been registered yet.] ([RS_CM_00218](#))

7.7.6 Find service

For the find service C++ API reference, see chapter [8.1.3.9](#).

[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](#))

7.7.7 Receive events

For the event data access C++ API reference, see chapter [8.1.3.13](#).

[SWS_CM_00709]{DRAFT} 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 [The sending of an event on sender side shall not lead to an implicit context switch to the receiver process, unless the receiver explicitly enabled it by following [[SWS_CM_00182](#)] and [[SWS_CM_00711](#)].] ([RS_CM_00203](#))

7.7.7.1 Receive event by polling

For the polling access no additional APIs on top of [8.1.3.13](#) are needed.

7.7.7.2 Receive event by getting triggered

For the receive event by getting triggered C++ API reference, see chapter [8.1.3.14](#).

[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](#))

7.7.8 Call a service method

For the call a service method C++ API reference, see chapter [8.1.3.15](#).

[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](#))

7.7.9 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)

7.7.10 Instance Specifier Translation

For the instance specifier translation C++ API reference, see chapter 8.1.3.18.

[SWS_CM_10452]{DRAFT} InstanceSpecifier translation to InstanceIdentifiers
[The Communication Management shall translate an `InstanceSpecifier` to `InstanceIdentifiers`. Based on the match there shall be zero, 1 or multiple `InstanceIdentifiers`.] (RS_CM_00207, RS_AP_00137)

7.7.11 Invalid Value

[SWS_CM_10453]{DRAFT} Implementation of `invalidValue` [For AUTOSAR data types which have an `invalidValue` specified, the Service Proxy shall provide the generic static `constexpr` field:

```
constexpr static <DataType> InvalidValue<DataType> = <InvalidValue>;
```

where

- `<DataType>` is the short name of the data type
- `InvalidValue<DataType>` is the field name that contains `InvalidValue` for `DataType`.
- `<InvalidValue>` is the value defined as `invalidValue` for the data type

] (RS_CM_00001)

Note: This requirement also applies to events.

8 Communication API specification

The adaptive platform supports multiple language bindings. At the current state only the C++ language binding is implemented.

8.1 C++ language binding

8.1.1 API Header files

This chapter describes the header files of the ara::com API.

The so-called *input* for the header files are the AUTOSAR metamodel classes within the *ServiceInterface* description, as defined in the AUTOSAR Adaptive Methodology Specification [25].

The following requirements are applicable for all header files; requirements which are specific for a header file are described in own sub-chapters.

The required folder structure for the ARA public header files is defined by [SWS_AP_00001] in AUTOSAR SWS General [26]. This applies to the *Types header file*, but the folder structure for the *Service header files*, *Common header files*, and the *Implementation Types header files* is derived from the namespace hierarchy.

[SWS_CM_01020]{DRAFT} Folder structure [The *Service header files* defined by [SWS_CM_01002], the *Common header files* defined by [SWS_CM_01012], and the *Implementation Types header files* defined by [SWS_CM_10373] 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] and [SWS_CM_10375].] (RS_CM_00001, RS_AP_00113, RS_AP_00114)

8.1.1.1 Service header files

The *Service header files* 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_00113](#), [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*:

```
1 #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 [\[SWS_CM_10375\]](#). <name> is the the `ServiceInterface.shortName` converted to lower-case letters.] ([RS_CM_00001](#), [RS_AP_00113](#), [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:

```
1 namespace <ServiceInterface.namespace[0].symbol> {
2 namespace <ServiceInterface.namespace[1].symbol> {
3 namespace <...> {
4 namespace <ServiceInterface.namespace[n].symbol> {
5 ...
6 } // namespace <ServiceInterface.namespace[n].symbol>
7 } // namespace <...>
8 } // namespace <ServiceInterface.namespace[1].symbol>
9 } // namespace <ServiceInterface.namespace[0].symbol>
```

with all namespace names converted to lower-case letters.] ([RS_CM_00002](#), [RS_AP_00113](#), [RS_AP_00114](#))

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:

```
1 namespace skeleton {
2 ...
3 } // namespace skeleton
```

] ([RS_CM_00002](#), [RS_AP_00113](#), [RS_AP_00114](#))

[SWS_CM_01007]{DRAFT} Service proxy namespace [The C++ namespace for a specific service proxy class shall be:

```
1 namespace proxy {
2 ...
3 } // namespace proxy
```

]([RS_CM_00002](#), [RS_AP_00113](#), [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:

```
1 namespace events {  
2 ...  
3 } // namespace events
```

]([RS_AP_00113](#), [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:

```
1 namespace methods {  
2 ...  
3 } // namespace methods
```

]([RS_CM_00002](#), [RS_AP_00113](#), [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:

```
1 namespace fields {  
2 ...  
3 } // namespace fields
```

]([RS_CM_00002](#), [RS_CM_00216](#), [RS_AP_00113](#), [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:

```
1 namespace <ServiceInterface.namespace[0].symbol> {  
2 namespace <ServiceInterface.namespace[1].symbol> {  
3 namespace <...> {  
4 namespace <ServiceInterface.namespace[n].symbol> {  
5 namespace skeleton {  
6 ...  
7 namespace events {  
8 ...  
9 } // namespace events  
10  
11 namespace methods {  
12 ...  
13 } // namespace methods  
14  
15 namespace fields {  
16 ...  
17 } // namespace fields  
18  
19 ...  
20 } // namespace skeleton
```

```

21 } // namespace <ServiceInterface.namespace[n].symbol>
22 } // namespace <...>
23 } // namespace <ServiceInterface.namespace[1].symbol>
24 } // 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:

```

1 namespace <ServiceInterface.namespace[0].symbol> {
2 namespace <ServiceInterface.namespace[1].symbol> {
3 namespace <...> {
4 namespace <ServiceInterface.namespace[n].symbol> {
5 namespace proxy {
6
7 namespace events {
8 ...
9 } // namespace events
10
11 namespace methods {
12 ...
13 } // namespace methods
14
15 namespace fields {
16 ...
17 } // namespace fields
18
19 ...
20 } // namespace proxy
21 } // namespace <ServiceInterface.namespace[n].symbol>
22 } // namespace <...>
23 } // namespace <ServiceInterface.namespace[1].symbol>
24 } // 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 *ApApplicationErrors* 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_00113*, *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*:

```

1 #include "ara/com/types.h"

```

]([RS_CM_00001](#), [RS_AP_00113](#), [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 [Cp-ImplementationDataType](#)s that are actually used by the particular [ServiceInterface](#):

```
1 #include "<namespace[0]>/<namespace[1]>/.../<namespace[n]>/impl_type_<  
symbol>.h"
```

where <namespace[0..n]> is the namespace hierarchy defined in [\[SWS_CM_10375\]](#), and <symbol> is the [Cpp Implementation Data Type symbol](#) according to section [8.1.2.5.2](#) converted to lower-case letters.]([RS_CM_00001](#), [RS_AP_00113](#), [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 [ApApplicationError](#)-Domains for the [ApApplicationError](#)s 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_00113](#), [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:

```
1 namespace ara {  
2     namespace com {  
3         ...  
4     } // namespace com
```

```
5 } // namespace ara
](RS_CM_00002, RS_AP_00113, 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

The *Implementation Types header files* include the ara::com specific type declarations derived from the `CppImplementationDataType`s created from the definitions of AUTOSAR meta model classes within the `ServiceInterface` description. Such data type declarations are described in detail in chapter 8.1.2.5.

[SWS_CM_10373]{DRAFT} Implementation Types header files existence [The communication management shall provide an *Implementation Types header file* for each `CppImplementationDataType` defined in the input by using the file name `impl_type_<symbol>.h`, where `<symbol>` is the `Cpp Implementation Data Type symbol` according to section 8.1.2.5.2 converted to lower-case letters.] (RS_CM_00001, RS_AP_00116)

The *Implementation Types header files* might need to include other header files, e.g. for `ara::core::String` or `ara::core::Vector`.

[SWS_CM_10374]{DRAFT} Data Type definitions for AUTOSAR Data Types in Implementation Types header files [The *Implementation Types header files* shall include the type definitions and structure and class definitions for all the AUTOSAR Data Types according to [SWS_CM_00402], [SWS_CM_00403], [SWS_CM_00404], [SWS_CM_00405], [SWS_CM_00406], [SWS_CM_00407], [SWS_CM_00408], [SWS_CM_00409], [SWS_CM_00410] and [SWS_CM_00424].] (RS_CM_00001)

[SWS_CM_10375]{DRAFT} Implementation Types header file namespace [The C++ namespace of the *Implementation Types header file* for a given `CppImplementationDataType` is defined via the aggregated `namespace`. Based on the `symbol` attributes of the ordered `SymbolProps` aggregated by `CppImplementationDataType` in role `namespace`, the C++ namespace of the *Implementation Types header file* shall be:

```
1 namespace <CppImplementationDataType.namespace[0].symbol> {
2 namespace <CppImplementationDataType.namespace[1].symbol> {
3 namespace <...> {
```

```
4 namespace <CppImplementationDataType.namespace[n].symbol> {  
5 ...  
6 } // namespace <CppImplementationDataType.namespace[n].symbol>  
7 } // namespace <...>  
8 } // namespace <CppImplementationDataType.namespace[1].symbol>  
9 } // namespace <CppImplementationDataType.namespace[0].symbol>
```

with all namespace names converted to lower-case letters.] ([RS_CM_00002](#), [RS_AP_00113](#), [RS_AP_00114](#))

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]{DRAFT} Raw data stream header file existence [The communication management shall provide a *Raw data stream header file* by using the file name `raw_stream.h`.] ([RS_CM_00001](#))

[SWS_CM_10489]{DRAFT} Raw data stream header file namespace [The C++ namespace for the data type definitions included by the *Raw data stream header file* shall be:

```
1 namespace ara {  
2 namespace com {  
3 namespace raw {  
4 ...  
5 } // namespace raw  
6 } // namespace com  
7 } // namespace ara
```

] ([RS_CM_00002](#))

[SWS_CM_10490]{DRAFT} 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 Metamodel.

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, Service Version Classes 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 version (`ServiceVersion`) and 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 types of `ServiceIdentifier` and `ServiceVersion` are specific to the Communication Management software provider. Their concrete realization is implementation defined. To allow for logging and for storing and managing in C++ container classes by the using application, however, the types of both classes 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 [27]. These requirements are fulfilled if the operators `operator==`, `operator<`, and `operator=` as well as a `toString()` method is provided.

```
1 class <ServiceInterface.shortName> {
2     public:
3         static constexpr ServiceIdentifierType ServiceIdentifier;
4         static constexpr ServiceVersionType ServiceVersion;
5
6         static std::uint32_t ServiceContractVersionMajor;
7         static std::uint32_t ServiceContractVersionMinor;
8     };
9
10 class ServiceIdentifierType {
```

```
11 bool operator==(const ServiceIdentifierType& other) const;
12 bool operator<(const ServiceIdentifierType& other) const;
13 ServiceIdentifierType& operator=(const ServiceIdentifierType& other);
14 ara::core::string_view ToString() const;
15 };
16
17 class ServiceVersionType {
18     bool operator==(const ServiceVersionType& other) const;
19     bool operator<(const ServiceVersionType& other) const;
20     ServiceVersionType& operator=(const ServiceVersionType& other);
21     ara::core::string_view ToString() const;
22 };
```

]([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_00350]{DRAFT} **Instance Specifier Class** [The `InstanceSpecifier` class is specified in [16].]([RS_CM_00101](#), [RS_AP_00137](#))

[SWS_CM_00302]{DRAFT} **Instance Identifier Class** [The Communication Management shall provide a class `InstanceIdentifier`. It only contains instance information, but does not contain a fully qualified name, which would also have service type information.

The definition of the `InstanceIdentifier` can be extended by the Communication Management software provider, but at least the given 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 [27] to allow for logging of `InstanceIdentifiers` as well as storing and managing `InstanceIdentifiers` 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.

```
1 class InstanceIdentifier {
2 public:
3
4     explicit InstanceIdentifier(ara::core::string_view value);
5     ara::core::string_view ToString() const;
6     bool operator==(const InstanceIdentifier& other) const;
```

```
7  bool operator<(const InstanceIdentifier& other) const;
8  InstanceIdentifier& operator=(const InstanceIdentifier& other);
9 };
```

]([RS_CM_00101](#), [RS_CM_00102](#), [RS_AP_00113](#), [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 [27]. A `ara::core::Vector` for example fulfills these requirements.]

```
1 using InstanceIdentifierContainer = ara::core::Vector<InstanceIdentifier>;
2
3 ](RS\_CM\_00101, RS\_CM\_00102, RS\_AP\_00113, 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 [27] to allow storing and managing `FindServiceHandles` 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:

```
1 struct FindServiceHandle {
2     internal::ServiceId service_id;
3     internal::InstanceId instance_id;
4     std::uint32_t uid;
5
6     bool operator==(const FindServiceHandle& other) const;
7     bool operator<(const FindServiceHandle& other) const;
8     FindServiceHandle& operator=(const FindServiceHandle& other);
9     ...
10 };
```

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 *handles*. 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 [27]. 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 `HandleTypes` 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.

```
1 class HandleType {  
2 public:  
3     bool operator==(const HandleType& other) const;  
4     bool operator<(const HandleType& other) const;  
5     const ara::com::InstanceIdentifier& GetInstanceId() const;  
6 };
```

](RS_CM_00102, RS_AP_00113, 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_00113, RS_AP_00114)

[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_00113, RS_AP_00114)

[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 [27]. A `ara::core::Vector` for example fulfills these requirements.]

```
1 template <typename T>
2 using ServiceHandleContainer = ara::core::Vector<T>;
```

]([RS_CM_00102](#), [RS_AP_00113](#), [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 must 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`.]

```
1 template <typename T>
2 using FindServiceHandler =
3 std::function<void(ServiceHandleContainer<T>, FindServiceHandle)>;
```

]([RS_CM_00102](#), [RS_AP_00113](#), [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 Pointers* 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(nullptr_t) noexcept;

    // Copy constructor is deleted
    SamplePtr ( const SamplePtr& ) = delete;

    // Move constructor
    SamplePtr( SamplePtr&& ) noexcept;

    // Default copy assignment operator is deleted
    SamplePtr& operator=( const SamplePtr& ) = delete;
    // Assignment of nullptr_t
    SamplePtr& operator=(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 noexcept;

    // Swaps the managed object
    void Swap ( SamplePtr& ) noexcept;
    // Replaces the managed object
    void Reset (nullptr_t) ;

    // Returns the stored object
    T* Get () const noexcept;

    // Returns the end 2 end protection check result
    ars::com::e2e::ProfileCheckStatus GetProfileCheckStatus() const noexcept;

};
```

](*RS_CM_00202, RS_CM_00203, RS_AP_00113, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135*)

[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:

```
1 ara::com::e2e::ProfileCheckStatus GetProfileCheckStatus() const noexcept;
2
```

](*RS_E2E_08534, RS_AP_00113, 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.

```
1 template <typename T>
2 using SampleAllocateePtr = std::unique_ptr<T>;
```

](*RS_CM_00201, RS_AP_00113, 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 a 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.

```
1 using EventReceiveHandler = std::function<void()>;
```

](*RS_AP_00113, 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.

```
1 enum class SubscriptionState : uint8_t {  
2     kSubscribed,  
3     kNotSubscribed,  
4     kSubscriptionPending  
5 };
```

](RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_AP_00113, 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.

```
1 using SubscriptionStateChangeHandler =  
2 std::function<void(SubscriptionState)>;
```

](RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_AP_00113, RS_AP_00114, RS_AP_00120)

8.1.2.3 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.

```
1 enum class MethodCallProcessingMode : uint8_t {  
2     kPoll,  
3     kEvent,  
4     kEventSingleThread  
5 };
```

](RS_CM_00211, RS_AP_00113, RS_AP_00114, RS_AP_00125)

The expected behavior of each processing mode is described in [SWS_CM_00198].

8.1.2.4 Generic Data Types

8.1.2.4.1 Future and Promise

The `Future` and `Promise` class templates are described in [16].

8.1.2.4.2 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].

8.1.2.4.3 Variant Data Types

The class template `ara::core::Variant` is used to provide a type-safe union representation is described in [16]. Whenever there is a mention of the standard C++17 Item `std::variant`, the implied source material is [28].

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-safe 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();

    // 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_00113, RS_AP_00114, RS_AP_-00122, RS_AP_00132, RS_AP_00127)

[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_00113, RS_AP_00114)

[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_00113, RS_AP_00114, RS_AP_-00132)

[SWS_CM_01053]{DRAFT} Variant copy constructor [The `Variant` copy constructor

```
1 Variant( const Variant& );
```

behaves as the `std::variant` copy constructor

```
1 constexpr variant( const variant& other );
```

](RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114)

[SWS_CM_01054]{DRAFT} Variant converting constructor [The `Variant` converting constructor

```
1 template< class T >
2 Variant ( T&& ) noexcept;
```

behaves as the `std::variant` converting constructor

```
1 template< class T >
2 constexpr variant( TT& t ) noexcept;
```

](RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114, RS_AP_00132)

[SWS_CM_01055]{DRAFT} Variant explicit converting constructor with specified alternative [The `Variant` explicit converting constructor with specified alternative

```
1 template< class T, class... Args >
2 explicit Variant ( std::in_place_type_t<T> , Arg&&... );
```

behaves as the `std::variant` explicit converting constructor with specified alternative

```
1 template< class T, class... Args >
2 constexpr explicit variant ( std::in_place_type_t<T> , Arg&&... args );
```

](RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114, RS_AP_00132)

[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

```
1 template< class T, class U, class... Args >
2 explicit Variant ( std::in_place_type_t<T> , std::initializer_list<U> ,
3 Arg&&... );
```

behaves as the `std::variant` explicit converting constructor with specified alternative and initializer list

```
1 template< class T, class U, class... Args >
2 constexpr explicit variant ( std::in_place_type_t<T> , std::
3 initializer_list<U> il, Arg&&... args );
```

](*RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114*)

[SWS_CM_01057]{DRAFT} Variant explicit converting constructor with alternative specified by index [The **Variant** explicit converting constructor with alternative specified by index

```
1 template< std::size_t I, class... Args >
2 explicit Variant ( std::in_place_index_t<I> , Arg&&... );
```

behaves as the `std::variant` with alternative specified by index

```
1 template< std::size_t I, class... Args >
2 constexpr explicit variant ( std::in_place_index_t<I> , Arg&&... args )
    ;
```

](*RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114*)

[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

```
1 template< std::size_t I, class U, class... Args >
2 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

```
1 template< std::size_t I, class U, class... Args >
2 constexpr explicit variant ( std::in_place_index_t<I> , std:::
    initializer_list<U> il, Arg&&... args );
```

](*RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114*)

[SWS_CM_01059]{DRAFT} Variant destructor [The **Variant** destructor

```
1 ~Variant();
```

behaves as the `std::variant` destructor

```
1 ~variant();
```

](*RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114*)

[SWS_CM_01060]{DRAFT} Variant move assignment operator [The **Variant** move assignment operator

```
1 Variant& operator=( Variant&& ) noexcept;
```

behaves as the `std::variant` move assignment operator

```
1 constexpr variant( variant&& rhs ) noexcept
```

](*RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114, RS_AP_-00132*)

[SWS_CM_01061]{DRAFT} Variant default copy assignment operator [The **Variant** default copy assignment operator

```
1 Variant& operator=(const Variant&);
```

behaves as the `std::variant` default copy assignment operator

```
1 variant& operator=( const variant& rhs );
```

[RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114]

[SWS_CM_01062]{DRAFT} Variant converting assignment operator [The `Variant` converting assignment operator

```
1 template < class T >
2 Variant& operator=( T&& ) noexcept;
```

behaves as the `std::variant` converting assignment operator

```
1 template < class T >
2 variant& operator=( T&& t ) noexcept;
```

[RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, 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

```
1 std::size_t index();
```

behaves as the `std::variant` function to return the zero-based index of the alternative

```
1 constexpr std::size_t index();
```

[RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, 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

```
1 bool valueless_by_exception() const noexcept;
```

behaves as the `std::variant` function to return false if the variant holds a value, else true

```
1 constexpr bool valueless_by_exception() const noexcept;
```

[RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, 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

```
1 template < class T, class... Args >
2 void emplace( Args&&... );
```

behaves as the `std::variant` `emplace` function to create a new value in-place, in an existing Variant object

```
1 template < class T, class... Args >
2 void emplace( Args&&... args );  
  
](RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, 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]

```
1 template < class T, class U, class... Args >
2 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

```
1 template < class T, class U, class... Args >
2 void emplace( std::initializer_list<U> il , Args&&... args );  
  
](RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_01068]{DRAFT} Variant function to create a new value in-place, in an existing Variant object by destroying 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]

```
1 template < std::size_t I, class... Args >
2 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

```
1 template < std::size_t I, class... Args >
2 void emplace( Args&&... args );  
  
The behavior is undefined if I is not less than sizeof...(Types)](RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_01069]{DRAFT} Variant function to create a new value in-place, in an existing Variant object by destroying 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]

```
1 template <size_t I, class U, class... Args>
2 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

```
1 template <size_t I, class U, class... Args>
2 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_00113, RS_AP_00114)
```

[SWS_CM_01065]{DRAFT} Variant function to swap two Variants [The Variant function swaps two Variants]

```
1 void swap( Variant& ) noexcept;
```

behaves as the std::variant function to swap two Variants

```
1 void swap( Variant& rhs ) noexcept;
```

](*RS_CM_00205, RS_SOMEIP_00050, RS_AP_00113, RS_AP_00114, RS_AP_00132*)

8.1.2.4.4 Scale Linear And Texttable Data Types

The following section describes the `ScaleLinearAndTexttable` class template used in `ara::com`. The objects of this class at a given time either hold the value of an enumerator of a specific enum class or other values of the underlying type of this. The used enum class is specified through a template argument.

[SWS_CM_10392]{DRAFT} ScaleLinearAndTexttable Class Template [The Communication Management shall at least provide an `ScaleLinearAndTexttable` class template that as described below:

```
template <typename T>
class ScaleLinearAndTexttable
{
public:
    // Declaration of the underlying_type
    static_assert(std::is_enum<T>::value, "Type T has to be an enum");
    using underlying_type = typename std::underlying_type<T>::type;

    // Default constructor
    explicit ScaleLinearAndTexttable();
    // Copy constructor
    explicit ScaleLinearAndTexttable(const ScaleLinearAndTexttable &v);
    // Constructing an object from an enum
    explicit ScaleLinearAndTexttable(const T &v);
    // Constructing an object from the underlying type of an enum
    explicit ScaleLinearAndTexttable(const underlying_type &v);

    // Copy assignment operator
    ScaleLinearAndTexttable& operator=(const ScaleLinearAndTexttable &v);
    // Assignment operator from an enum
    ScaleLinearAndTexttable& operator=(const T &v);
    // Assignment operator from the underlying type of an enum
    ScaleLinearAndTexttable& operator=(const underlying_type &v);
    // Casting operator to the underlying_type
    explicit operator underlying_type() const;

    // Equal to operator to another ScaleLinearAndTexttable<T>
```

```
friend bool operator==(const ScaleLinearAndTexttable<T> &lhs,
                      const ScaleLinearAndTexttable<T> &rhs);
// Equal to operator to the underlying_type
friend bool operator==(const ScaleLinearAndTexttable<T> &lhs,
                      const underlying_type &rhs);
// Equal to operator to the underlying_type
friend bool operator==(const underlying_type &lhs,
                      const ScaleLinearAndTexttable<T> &rhs);
// Equal to operator to the enum
friend bool operator==(const ScaleLinearAndTexttable<T> &lhs,
                      const T &rhs);
// Equal to operator to the enum
friend bool operator==(const T &lhs,
                      const ScaleLinearAndTexttable<T> &rhs);

// Not-equal to operator to another ScaleLinearAndTexttable<T>
friend bool operator!=(const ScaleLinearAndTexttable<T> &lhs,
                      const ScaleLinearAndTexttable<T> &rhs);
// Not-equal to operator to the underlying_type
friend bool operator!=(const ScaleLinearAndTexttable<T> &lhs,
                      const underlying_type &rhs);
// Not-equal to operator to the underlying_type
friend bool operator!=(const underlying_type &lhs,
                      const ScaleLinearAndTexttable<T> &rhs);
// Not-equal to operator to the enum
friend bool operator!=(const ScaleLinearAndTexttable<T> &lhs,
                      const T &rhs);
// Not-equal to operator to the enum
friend bool operator!=(const T &lhs,
                      const ScaleLinearAndTexttable<T> &rhs);
};

](RS\_CM\_00003, RS\_AP\_00113, RS\_AP\_00114, RS\_AP\_00122)
```

[SWS_CM_10393]{DRAFT} **ScaleLinearAndTexttable static assertion** [The **ScaleLinearAndTexttable** shall check whether the **T** template argument is an enum type.

```
1 static_assert(std::is_enum<T>::value, "Type_T_has_to_be_an_enum");
```

Rationale: The `std::underlying_type<T>` in **[SWS_CM_10394]** has an undefined behavior for non-enum inputs.]([RS_CM_00003](#), [RS_AP_00113](#), [RS_AP_00114](#))

[SWS_CM_10394]{DRAFT} **ScaleLinearAndTexttable underlying type deduction** [The **ScaleLinearAndTexttable** shall deduct and store the underlying type of the enum it was defined with.

```
1 using underlying_type = typename std::underlying_type<T>::type;
```

Rationale: The **ScaleLinearAndTexttable** is designed to hold values of this type.]([RS_CM_00003](#), [RS_AP_00113](#), [RS_AP_00114](#))

[SWS_CM_10395]{DRAFT} `ScaleLinearAndTexttable` default constructor [The `ScaleLinearAndTexttable` shall have a default constructor with the following declaration:

```
  explicit ScaleLinearAndTexttable();  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10396]{DRAFT} `ScaleLinearAndTexttable` copy constructor [The `ScaleLinearAndTexttable` shall have a copy constructor with the following declaration:

```
  explicit ScaleLinearAndTexttable(const ScaleLinearAndTexttable &v);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10397]{DRAFT} `ScaleLinearAndTexttable` constructor with enum class argument [The `ScaleLinearAndTexttable` shall have a constructor with the same argument as the enum class that was given as the T template parameter with the following declaration:

```
  explicit ScaleLinearAndTexttable(const T &v);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10398]{DRAFT} `ScaleLinearAndTexttable` constructor with underlying type argument [The `ScaleLinearAndTexttable` shall have a constructor with the same argument that was deducted from the T template parameter with the following declaration:

```
  explicit ScaleLinearAndTexttable(const underlying_type &v);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10399]{DRAFT} `ScaleLinearAndTexttable` copy assignment operator [The `ScaleLinearAndTexttable` shall have a copy assignment operator with the following declaration:

```
  ScaleLinearAndTexttable& operator=(const ScaleLinearAndTexttable &v);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10400]{DRAFT} `ScaleLinearAndTexttable` assignment operator with enum class argument [The `ScaleLinearAndTexttable` shall have an assignment operator with the same argument as the enum class that was given as the T template parameter with the following declaration:

```
  ScaleLinearAndTexttable& operator=(const T &v);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10401]{DRAFT} `ScaleLinearAndTexttable` assignment operator with underlying type argument [The `ScaleLinearAndTexttable` shall have an assignment operator with the same argument that was deducted from the T template parameter with the following declaration:

```
1 ScaleLinearAndTexttable& operator=(const underlying_type &v);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10402]{DRAFT} `ScaleLinearAndTexttable` cast operator to the underlying type [The `ScaleLinearAndTexttable` shall have a cast operator to the underlying type that was deducted from the T template parameter with the following declaration:

```
1 explicit operator underlying_type() const;  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10403]{DRAFT} Equal to operator between two `ScaleLinearAndTexttable` objects [The `ScaleLinearAndTexttable` shall have an equal to operator to compare two `ScaleLinearAndTexttable` objects with the following declaration:

```
1 friend bool operator==(const ScaleLinearAndTexttable<T> &lhs,  
2 const ScaleLinearAndTexttable<T> &rhs);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10404]{DRAFT} Equal to operators between `ScaleLinearAndTexttable` and an underlying type [The `ScaleLinearAndTexttable` shall have equal to operators to compare a `ScaleLinearAndTexttable` object to the the underlying type that was deducted from the T template parameter with the following declarations:

```
1 friend bool operator==(const ScaleLinearAndTexttable<T> &lhs,  
2 const underlying_type &rhs);  
3 friend bool operator==(const underlying_type &lhs,  
4 const ScaleLinearAndTexttable<T> &rhs);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10405]{DRAFT} Equal to operators between `ScaleLinearAndTexttables` and an enum class [The `ScaleLinearAndTexttable` shall have equal to operators to compare a `ScaleLinearAndTexttable` object to the same enum class that was given as the T template parameter with the following declarations:

```
1 friend bool operator==(const ScaleLinearAndTexttable<T> &lhs,  
2 const T &rhs);  
3 friend bool operator==(const T &lhs,  
4 const ScaleLinearAndTexttable<T> &rhs);  
](RS_CM_00003, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10406]{DRAFT} Not equal to operator between two `ScaleLinearAndTexttable` objects [The `ScaleLinearAndTexttable` shall have a not equal to operator to compare two `ScaleLinearAndTexttable` objects with the following declaration:

```
1 friend bool operator!=(const ScaleLinearAndTexttable<T> &lhs,  
2 const ScaleLinearAndTexttable<T> &rhs);
```

](*RS_CM_00003, RS_AP_00113, RS_AP_00114*)

[SWS_CM_10407]{DRAFT} Not equal to operators between `ScaleLinearAndTexttable` and an underlying type [The `ScaleLinearAndTexttable` shall have not equal to operators to compare an `ScaleLinearAndTexttable` object to the underlying type that was deducted from the T template parameter with the following declarations:

```
1 friend bool operator!=(const ScaleLinearAndTexttable<T> &lhs,
2                               const underlying_type &rhs);
3 friend bool operator!=(const underlying_type &lhs,
4                               const ScaleLinearAndTexttable<T> &rhs);
```

](*RS_CM_00003, RS_AP_00113, RS_AP_00114*)

[SWS_CM_10408]{DRAFT} Not equal to operators between `ScaleLinearAndTexttables` and an enum class [The `ScaleLinearAndTexttable` shall have not equal to operators to compare a `ScaleLinearAndTexttable` object to the same enum class that was given as the T template parameter with the following declarations:

```
1 friend bool operator!=(const ScaleLinearAndTexttable<T> &lhs,
2                               const T &rhs);
3 friend bool operator!=(const T &lhs,
4                               const ScaleLinearAndTexttable<T> &rhs);
```

](*RS_CM_00003, RS_AP_00113, RS_AP_00114*)

8.1.2.5 Communication Payload Data Types

The data types described in the previous chapters are derived from the ara::com API design and as an integral part of the API, they explicitly need to exist to make use of ara::com API.

In contrast to this, the types described in this chapter will exist only if there is a related `AutosarDataType` configured by the user, i.e. they are fully dependent to the data type related input configuration. These data types are intended to be used for the definition of the "payload" of events, operations, fields, and errors but also for the implementation of the ara::com API and the functionality of the Adaptive Applications.

The parameters used in the event, method signatures, and errors of the ara::com API are depending on the design of the service. So they are usually generated based on the `DataPrototypes` of the `ServiceInterface` description. Their mapping to C++ data types is described in following.

The AUTOSAR Meta Model defines the `AutosarDataPrototype` which can be typed by an `ApplicationDataType` or an `CppImplementationDataType`, but the Communication Management maps only `CppImplementationDataTypes` to C++ data types. Therefore it is required in the input configuration that every `Application-DataType` used for the typing of a `DataPrototype` is mapped by a `DataTypeMap` to an `CppImplementationDataType`.

The `PortInterfaceToDataTypeMapping` associates a particular `ServiceInterface` with a `DataTypeMappingSet` and defines thus the applicable `DataTypeMaps`.

[SWS_CM_00423]{DRAFT} Data Type Mapping [The ARA generator shall reject input configurations containing a `AutosarDataPrototype` which is typed by an `ApplicationDataType`, but not mapped to an `CppImplementationDataType`.] (*RS_CM_00211, RS_CM_00003*)

The *Implementation Types header files* as defined in [SWS_CM_10373] includes the type declarations derived from the `CppType`s of the *AUTOSAR Adaptive Platform* meta-model classes, depending on the value of the `typeEmitter` attribute (see [TPS_MANI_01176] and [TPS_MANI_01177] of the AUTOSAR Manifest Specification [5]).

[SWS_CM_00421]{DRAFT} Provide data type definitions [The ARA generator shall provide the corresponding data type definition according to the rules defined in [TPS_MANI_01176] and [TPS_MANI_01177] of the AUTOSAR Manifest Specification [5].] (*RS_CM_00211, RS_CM_00003*)

The redeclaration of C++ types due to the multiple descriptions of equivalent `CppType`s in the `ServiceInterface` description shall be avoided.

[SWS_CM_00411]{DRAFT} Avoid Data Type redeclaration [If there are several data types with equal `CppType`s defined which are referring to compatible `CppType`s with identical `CppType`s, there shall exist only one corresponding type declaration as described in the following sub chapters.] (*RS_CM_00211, RS_CM_00003*)

The available meta-model classes are described in detail in the AUTOSAR Manifest Specification [5].

8.1.2.5.1 Classification of Cpp Implementation Data Types

The type model `CppType` is able to express following kinds of data types:

- Primitive Cpp Implementation Data Type
- Array Cpp Implementation Data Type
- Structure Cpp Implementation Data Type
- Variant Cpp Implementation Data Type
- String Cpp Implementation Data Type
- Vector Cpp Implementation Data Type
- Associative Map Cpp Implementation Data Type

- Redefinition Cpp Implementation Data Type
- Enumeration Data Type
- Scale Linear And Texttable Data Type

A Primitive Cpp Implementation Data Type is classified by the `category` attribute set to `VALUE`. Please note that the usage of the `category` `VALUE` is restricted to `StdCppImplementationDataTypes` according to [constr_1578] defined in [5].

An Array Cpp Implementation Data Type is classified by the `category` attribute set to `ARRAY`. If the subclass `StdCppImplementationDataType` is used the array will be implemented as a `ara::core::Array`. The `StdCppImplementationDataType` of category `ARRAY` has one `templateArgument` that points with the `templateType` reference to the data type of elements that are contained in the array. The referenced `CppType` itself can be one of the listed kinds again. The array size is specified with the `arraySize`. If the subclass `CustomCppMethodImplementationDataType` is used the array will be implemented as a custom array that is declared in the `headerFile` of the `CustomCppMethodImplementationDataType`.

A Structure Cpp Implementation Data Type is classified by the `category` attribute of the `StdCppMethodImplementationDataType` set to `STRUCTURE` that has aggregated `CppMethodImplementationDataElement`s in the role `subElement`.

A Variant Cpp Implementation Data Type is classified by the `category` attribute of the `CppMethodImplementationDataType` set to `VARIANT`. A type alternative that is stored in a `CppMethodImplementationDataType` of category `VARIANT` is defined by an aggregated `templateArgument` that points with the `templateType` reference to the data type of the type alternative. If the subclass `StdCppMethodImplementationDataType` is used the variant will be implemented as `ara::core::Variant`. This template class is specified in [16]. If the subclass `CustomCppMethodImplementationDataType` is used the variant will be implemented as a custom variant that is declared in the `headerFile` of the `CustomCppMethodImplementationDataType`.

A String Cpp Implementation Data Type is classified by the `category` attribute of the `CppMethodImplementationDataType` set to `STRING`. Please note that the usage of the `category` `STRING` is restricted to `StdCppMethodImplementationDataTypes` according to [constr_1578] defined in [5].

A Vector Cpp Implementation Data Type is classified by the `category` attribute of the `CppMethodImplementationDataType` set to `VECTOR`. If the subclass `StdCppMethodImplementationDataType` is used the vector will be implemented as a `ara::core::Vector`. The `StdCppMethodImplementationDataType` of category `VECTOR` is allowed to have one `templateArgument` that points with the `templateType` reference to the data type of elements that are contained in the vector. The referenced `CppMethodImplementationDataElement` itself can be one of the listed kinds again. Optionally the `StdCppMethodImplementationDataType` of category `VECTOR` may have an additional `templateArgument` that defines the used `Allocator` with the `allocator` reference. If the subclass `CustomCppMethodImplementationDataType` is used

the vector will be implemented as a custom vector that is declared in the `headerFile` of the `CustomCppImplementationDataType`.

An Associative Map Cpp Implementation Data Type is classified by the `category` attribute of the `CppClassImplementationDataType` set to ASSOCIATIVE_MAP. If the subclass `StdCppImplementationDataType` is used the map will be implemented as a `ara::core::Map`. The `StdCppImplementationDataType` of category ASSOCIATIVE_MAP may have two or three `templateArguments`s. The first `templateArgument`s defines the key with the `templateType` reference, the second defines the value and the third defines the optional `Allocator` with the `allocator` reference. If the subclass `CustomCppImplementationDataType` is used the map will be implemented as a custom map that is declared in the `headerFile` of the `CustomCppImplementationDataType`.

A Redefinition Cpp Implementation Data Type is classified by the `category` attribute of the referring `StdCppImplementationDataType` set to TYPE_REFERENCE. The `StdCppImplementationDataType` with the category TYPE_REFERENCE points to an another `CppClassImplementationDataType` with the `typeReference` and defines a type alias in this way.

An Enumeration Data Type is classified by a Redefinition Cpp Implementation Data Type that boils down to a Primitive Cpp Implementation Data Type having a `SwDataDefProps` referencing a `CompuMethod`, where the `CompuMethod` has:

- the `category` attribute set to TEXTTABLE,
- and has a `CompuScales` container located in the `compuInternalToPhys` container,
- and the `CompuScales` container has `CompuScales` in role `compuScale` with point ranges only (i. e. lower and upper limit of a `CompuScale` are identical).

A Scale Linear And Texttable Data Type is classified by a Redefinition Cpp Implementation Data Type that boils down to a Primitive Cpp Implementation Data Type having a `SwDataDefProps` referencing a `CompuMethod`, where the `CompuMethod` has:

- the `category` attribute set to SCALE_LINEAR_AND_TEXTTABLE,
- and has a `CompuScales` container located in the `compuInternalToPhys` container,
- and the `CompuScales` container has `CompuScales` in role `compuScale` with point ranges (i. e. lower and upper limit of a `CompuScale` are identical) and non-point ranges where the `CompuRationalCoeffs` define a linear function

Please note that the usage of the different kinds of `CppClassImplementationDataType`s is described in more detail in the AUTOSAR Manifest Specification [5].

8.1.2.5.2 Naming of Implementation Data Types

The data type name is defined by the `Cpp Implementation Data Type` symbol, which is the `shortName` of the `CppClassImplementationDataType`.

[SWS_CM_00400]{DRAFT} Naming of data types by short name [The `CppClass Implementation Data Type` symbol shall be the `shortName` of the `CppClassImplementationDataType`.] (*RS_CM_00211, RS_CM_00003*)

8.1.2.5.3 Primitive Implementation Data Type

The Communication Management declares C++ types for all `Primitive Cpp Implementation Data Types` defined in the `ServiceInterface` that are classified by the `category` attribute set to `VALUE`. Please note that only `StdCppClassImplementationDataType` are allowed to have the `category` attribute set to `VALUE`.

[SWS_CM_00504]{DRAFT} Supported Primitive Cpp Implementation Data Types [The `StdCppClassImplementationDataType` with the `category` attribute set to `VALUE` is allowed to have one of the following `shortName`s:

- `int8_t`
- `int16_t`
- `int32_t`
- `int64_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`
- `uint64_t`
- `bool`
- `float`
- `double`

] (*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00122*)

Since only a defined set of `StdCppClassImplementationDataType`s with category `VALUE` is supported the primitive C++ datatypes `float`, `bool` and `double` are supported in addition to chosen fixed width integer types defined in the standard library header `<cstdint>`.

[SWS_CM_00402]{DRAFT} Primitive fixed width integer types [If a `StdCppImplementationDataType` with the category `VALUE` is defined in the `ServiceInterface` the standard library header `<cstdint>` shall be included if the `StdCppImplementationDataType` has one of the following Cpp Implementation Data Type symbols:

- `int8_t`
- `int16_t`
- `int32_t`
- `int64_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`
- `uint64_t`

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114*)

8.1.2.5.4 Array Implementation Data Type

The Communication Management declares C++ types for all `Array Cpp Implementation Data Types` defined in the `ServiceInterface`. In AUTOSAR Adaptive Platform, the C++ binding of an `Array Cpp Implementation Data Type` could either be implemented as an `ara::core::Array` or as a custom array.

An array definition is based on the following information:

- the array type,
- the number of dimensions,
- the number of elements for each dimension.

An `Array Cpp Implementation Data Type` can have one or multiple dimensions.

In the context of the definitions given in this chapter, the term *dimension* is not related to the real physical dimensions in the memory, but to the ostensible dimensions visible directly at the declaration of the data type. This means, that e.g. even if an `Array Cpp Implementation Data Type` holds elements of types different from `Array Cpp Implementation Data Type` which itself has array or vector elements, the term *one-dimensional* applies for the definition of the data type.

A *one-dimensional* `StdCppImplementationDataType` of category `ARRAY` aggregates exactly one `templateArgument` that defines the type of elements that are contained in the array with the `templateType` reference, e.g. in case of an

one-dimensional array of uint16 elements the `templateType` reference will point to a `Primitive Cpp Implementation Data Type` that represents the uint16 element. The array size is defined with the `arraySize` attribute.

[SWS_CM_00403]{DRAFT} `StdCppImplementationDataType` of category ARRAY with one dimension [For each `StdCppImplementationDataType` of category ARRAY with one dimension, there shall exist the corresponding type declaration as:

```
1 using <name> ara::core::Array<<element>, <size>>;
```

where:

`<name>` is the `CppClass Implementation Data Type` symbol of the `Array Cpp Implementation Data Type`,

`<element>` is the array element specification. It is defined by the `templateArgument` that refers to a `CppClassImplementationDataType` with the `templateType` reference.

- If the `CppClassTemplateArgument` is marked with `inplace = false` the `shortName` of the referenced `CppClassImplementationDataType` is used and the declaration of the referenced `CppClassImplementationDataType` is defined **outside** of the array.
- If the `CppClassTemplateArgument` is marked with `inplace = true` an unnamed `CppClassImplementationDataType` is defined as value type of the array and the `shortName` of the referenced `CppClassImplementationDataType` is ignored.

`<size>` is the defined `arraySize`.

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127*)

In case that a `StdCppImplementationDataType` with `category` ARRAY and the `shortName` `MyArray` has a `CppClassTemplateArgument` that points with the `templateType` reference to a `StdCppImplementationDataType` with `category` ARRAY and the `CppClassTemplateArgument` is marked with `inplace = true` this will result in the following code:

```
1 using MyArray = ara::core::Array<ara::core::Array<uint16_t, 10>, 5>;
```

If the `CppClassTemplateArgument` is marked with `inplace = false` this will result in the following code:

```
1 using MyInsideArray = ara::core::Array<uint16_t, 10>;
2 using MyArray = ara::core::Array<MyInsideArray, 5>;
```

A *multidimensional* `CppClassImplementationDataType` of `category` ARRAY contains nested `CppClassImplementationDataTypes` of `category` ARRAY. This means, that the `CppClassImplementationDataType` of `category` ARRAY will refer to a `CppClassImplementationDataType` of `category` ARRAY via the aggregated `templateArgument`. Such a definition describes a *two-dimensional* `Array Cpp Implementation Data Type`;

consequently a type with more dimensions is described by just nesting more `CppImplementationDataTypes` of category `ARRAY`. The innermost `CppImplementationDataTypes` of category `ARRAY` has the reference to the type of elements that are contained in the array.

[SWS_CM_00404]{DRAFT} Array Data Type with more than one dimension [For each `Array Cpp Implementation Data Type` having more than one dimension, there shall exist the corresponding type declaration according to [SWS_CM_00403] as base where `<element>` has a nested array for each additional dimension. The total number of dimensions is equal to the number of nested `CppImplementationDataTypes` with `category ARRAY` plus one for the top level `Array Cpp Implementation Data Type`. The array element itself is specified by the innermost `CppImplementationDataTypes` with `category` different from `ARRAY`.] ([RS_CM_00211](#), [RS_CM_00003](#), [RS_AP_00113](#), [RS_AP_00114](#))

Please note that [SWS_CM_00404] and a `StdCppImplementationDataType` with `category ARRAY` leads to an `ara::core::Array` type definition where the `<size>` definitions for each dimension are ordered from the leaf to the root `ImplementationDataTypeElement`, like e.g.:

```
1 using My2DimArray = <ara::core::Array<ara::core::Array<uint16, 3>, 2>;
```

which is the same layout as the corresponding C-style array type definition where the `<size>` definitions for each dimension are ordered from the root to the leaf, like:

```
1 typedef uint16 My2DimArray[2][3];
```

With the usage of `CustomCppMethodImplementationDataType` it is possible to specify a data type definition that is taken as the basis for a C++ language binding to a custom implementation that is declared in the configured `headerFile`. In case of a `CustomCppMethodImplementationDataType` the model defines the following:

- Class-Name of the custom implementation (`CustomCppMethodImplementationDataType.shortName`)
- Namespace of the custom implementation (`CustomCppMethodImplementationDataType.namespace`)
- Header File that contains the custom class declaration (`CustomCppMethodImplementationDataType.headerFile`).

[SWS_CM_00502]{DRAFT} CustomCppMethodImplementationDataType of category ARRAY [If a `CustomCppMethodImplementationDataType` of category `ARRAY` is used that contains a single `templateArgument` that refers to a `CppType` with the `templateType` reference and has the `arraySize` attribute set to a value the following type declaration shall be available in the included `headerFile`:

```
1 <ClassName><<element>, <size>>;
```

where:

`<ClassName>` is the `CppType` symbol of the `CustomCppMethodImplementationDataType` of category `ARRAY`. Please note that the

namespace that is defined with an ordered list of defined symbol is already handled by [SWS_CM_10375],

<element> is the array element specification. It is defined by the templateArgument that refers to the array element with the templateType reference.

<size> is the defined arraySize.

] (RS_AP_00113, RS_AP_00114)

Please note that multidimensional CustomCppImplementationDataTypes of category ARRAY are handled in the same way as StdCppImplementationDataTypes of category ARRAY. [SWS_CM_00404] is also valid for CustomCppImplementationDataTypes of category ARRAY.

8.1.2.5.5 Structure Implementation Data Type

The Communication Management declares C++ types for all Structure Cpp Implementation Data Types defined in the ServiceInterface.

[SWS_CM_00405]{DRAFT} **Structure Data Type** [For each Structure Cpp Implementation Data Type, there shall exist the corresponding type declaration as:

```
struct <name> {<elements>};
```

where:

<name> is the Cpp Implementation Data Type symbol of the Structure Cpp Implementation Data Type,

<elements> are record element specifications defined in Structure Cpp Implementation Data Type by ordered CppImplementationDataTypeElements. For each record element defined by one CppImplementationDataTypeElement one record element specification <elements> is defined. The record element specifications shall be ordered according to the order of the related CppImplementationDataTypeElements in the input configuration. Sequent record elements are separated with a semicolon.

] (RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114)

[SWS_CM_00414]{DRAFT} **Element specification typed by CppImplementationDataType** [Record element specifications <elements> of [SWS_CM_00405] shall exist as

```
<type> <name>;
```

where:

<type>

- is the Cpp Implementation Data Type symbol of the referred CppImplementationDataType if the typeReference is marked with `inplace = false`. In this case the type declaration of the referenced CppImplementationDataType is defined **outside** of the struct.
- is the type declaration of the referenced CppImplementationDataType if the typeReference is marked with `inplace = true`. In this case the type declaration is defined **inside** of the struct.

`<name>` is the `shortName` of the `ImplementationDataTypeElement`.

`] (RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114)`

If the `CppTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category ARRAY` and the `inplace` flag is set to **false** for the `typeReference` a using-declaration shall exist **outside** of the structure according to the rules defined in chapter 8.1.2.5.4.

```
1 struct Foo {  
2     MyArray elementX;  
3 };  
4  
5 using MyArray = ara::core::Array<uint8_t, 5>;
```

If the `CppTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category ARRAY` and the `inplace` flag is set to **true** for the `typeReference` an unnamed array shall be defined as member type of the struct and the `shortName` of the referenced `StdCppImplementationDataType` is ignored.

```
1 struct Foo {  
2     ara::core::Array<uint8_t, 5> elementX;  
3 };
```

If the `CppTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category VECTOR` and the `inplace` flag is set to **false** for the `typeReference` a using-declaration shall exist **outside** of the structure according to the rules defined in chapter 8.1.2.5.8.

If the `CppTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category VECTOR` and the `inplace` flag is set to **true** for the `typeReference` an unnamed vector shall be defined as member type of the struct and the `shortName` of the referenced `StdCppImplementationDataType` is ignored.

If the `CppTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category VARIANT` and the `inplace` flag is set to **false** for the `typeReference` a using-declaration shall exist **outside** of the structure according to the rules defined in chapter 8.1.2.5.6.

If the `CppTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category VARIANT` and the `inplace`

flag is set to **true** for the `typeReference` an unnamed variant shall be defined as member type of the struct and the `shortName` of the referenced `StdCppImplementationDataType` is ignored.

If the `CppClassImplementationDataTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category` ASSOCIATIVE_MAP and the `inplace` flag is set to **false** for the `typeReference` a using-declaration shall exist **outside** of the structure according to the rules defined in chapter 8.1.2.5.9.

If the `CppClassImplementationDataTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category` ASSOCIATIVE_MAP and the `inplace` flag is set to **true** for the `typeReference` an unnamed map shall be defined as member type of the struct and the `shortName` of the referenced `StdCppImplementationDataType` is ignored.

If the `CppClassImplementationDataTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category` STRUCTURE and the `inplace` flag is set to **false** for the `typeReference` a struct-declaration shall exist **outside** of the structure according to the rule defined in [SWS_CM_00405].

```
1 struct Foo {  
2     Bar elementX;  
3 };  
4  
5 struct Bar {  
6     ...  
7 };
```

If the `CppClassImplementationDataTypeElement` points with the `typeReference` to a `StdCppImplementationDataType` with `category` STRUCTURE and the `inplace` flag is set to **true** for the `typeReference` an unnamed struct shall be defined as member type of the struct and the `shortName` of the referenced `StdCppImplementationDataType` is ignored.

```
1 struct Foo {  
2     struct {  
3         ...  
4     } elementX;  
5 };
```

[SWS_CM_01032]{DRAFT} **Accessing optional record elements inside a Structure Cpp Implementation Data Type that are serialized with the Tag-Length-Value principle.** [For each record element inside a `Structure Cpp Implementation Data Type` which is marked as optional according to [TPS_MANI_01083], [TPS_MANI_01085] and [TPS_MANI_01084], there shall exist the corresponding type declaration as:

```
struct <struct name>{  
    ara::core::Optional<element datatype> <name>;  
}  
e.g.  
struct MyStruct {
```

```
    ara::core::Optional<bool> myBool;  
}
```

where:

<name> is the `shortName` of the optional `CppImplementationDataTypeElement`,

<element datatype>

- is the `shortName` of the referred `CppClassImplementationType` if the `typeReference` is marked with `inplace = false`. In this case the type declaration of the referenced `CppClassImplementationType` is defined **outside** of the struct.
- is the type declaration of the referenced `CppClassImplementationType` if the `typeReference` is marked with `inplace = true`. In this case the type declaration is defined **inside** of the struct.

|(RS_CM_00205, RS_SOMEIP_00050, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127) The template class `ara::core::Optional` is specified in [16].

8.1.2.5.6 Variant Implementation Data Type

The Communication Management declares C++ types for all `Variant Cpp Implementation Data Types` defined in the `ServiceInterface`.

[SWS_CM_00449]{DRAFT} Variant Data Type [For each Variant Cpp Implementation Data Type, there shall exist the corresponding type declaration as:

```
using <name> = ara::core::Variant< <elements> >;
```

where:

<name> is the `CppClassImplementationType symbol` of the `Variant Cpp Implementation Data Type`,

<elements> is the Variant element specification.

Each type alternative in a `StdCppClassImplementationType` of `category VARIANT` is defined with a `CppClassTemplateArgument` that points with the `templateType` reference to the `StdCppClassImplementationType` that represents the alternative. For each `CppClassTemplateArgument` one element specification <elements> is defined. The Variant element specifications are ordered according the order of the related `CppClassTemplateArguments`s in the input configuration. Sequent Variants elements are separated with a semicolon.

- If the `CppClassTemplateArgument` is marked with `inplace = false` the `shortName` of the referenced `CppClassImplementationDataType` is used and the declaration of the referenced `CppClassImplementationDataType` is defined **outside** of the variant.
- If the `CppClassTemplateArgument` is marked with `inplace = true` an unnamed `CppClassImplementationDataType` is defined as type that may be stored in this variant and the `shortName` of the referenced `CppClassImplementationDataType` is ignored.

] (*RS_CM_00211, RS_AP_00113, RS_AP_00114, RS_AP_00127*)

A Variant data type describes a kind of structural overlay. Defining only one element in a VARIANT is therefore not reasonable and indicates an error.

This template class is specified in paragraph 8.1.2.4.3.

[SWS_CM_00508]{DRAFT} `CustomCppClassImplementationDataType` of category VARIANT [If a `CustomCppClassImplementationDataType` of category VARIANT is used the following type declaration shall be available in the included `headerFile`:

1 <ClassName><<elements>>;

where:

<**ClassName**> is the Cpp Implementation Data Type symbol of the `CustomCppClassImplementationDataType` of category VARIANT. Please note that the namespace that is defined with an ordered list of defined symbol is already handled by **[SWS_CM_10375]**,

<**elements**> is the variant element specification. Each type alternative in a `CustomCppClassImplementationDataType` of category VARIANT is defined with a `CppClassTemplateArgument` that points with the `templateType` reference to the `CustomCppClassImplementationDataType` that represents the alternative. For each `CppClassTemplateArgument` one element specification <**elements**> is defined. The Variant element specifications are ordered according the order of the related `CppClassTemplateArgument`s in the input configuration. Sequent Variants elements are separated with a semicolon.

] (*RS_AP_00113, RS_AP_00114*)

8.1.2.5.7 String Implementation Data Type

The Communication Management declares C++ types for all `String Cpp Implementation Data Types` defined in the `ServiceInterface`.

[SWS_CM_00406]{DRAFT} `StdCppClassImplementationDataType` with the category STRING [For each `StdCppClassImplementationDataType` of category STRING there shall exist the corresponding type declaration as:

```
using <name> = ara::core::String;
```

where:

<name> is the Cpp Implementation Data Type symbol of the String Cpp Implementation Data Type.

] (RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127)

Please note that for the moment the C++ binding of a String Cpp Implementation Data Type is restricted to ara::core::String that is defined in [16].

Please note that optionally a custom Allocator is allowed to be defined as templateArgument for a String Cpp Implementation Data Type.

[SWS_CM_00509]{DRAFT} **StdCppImplementationDataType with the category STRING with a defined Allocator** If a StdCppImplementationDataType with the category STRING contains a templateArgument that points with the allocator reference to a custom Allocator the following type is declared:

```
using <name> = ara::core::BasicString<<allocator>>;
```

where:

<name> is the Cpp Implementation Data Type symbol of the String Cpp Implementation Data Type.

<allocator> is the <allocator namespace>::<allocator shortname> of the defined Allocator that is referenced by a CppTemplateArgument of String Cpp Implementation Data Type with the allocator reference,

] (RS_CM_00211, RS_CM_00003, RS_AP_00127)

8.1.2.5.8 Vector Implementation Data Type

The Communication Management declares C++ types for all Vector Cpp Implementation Data Types defined in the ServiceInterface. In AUTOSAR Adaptive Platform, the C++ binding of a Vector Cpp Implementation Data Type could either be implemented as an ara::core::Vector or as a custom vector.

A vector definition is based on the following information:

- the data type the vector consists of,
- the number of dimensions,
- optionally an Allocator that is used to acquire/release memory and to construct/destroy the elements in that memory.

A Vector Cpp Implementation Data Type can have one or multiple dimensions.

In the context of the definitions given in this chapter, the term *dimension* is used with the same sense as described in chapter 8.1.2.5.4.

A `CppClassImplementationDataType` of `category VECTOR` aggregates one `templateArgument` that defines the type of elements that are contained in the vector with the `templateType` reference, e.g. in case of an one-dimensional vector of `uint16` elements the `templateType` reference will point to a `Primitive Cpp Implementation Data Type` that represents the `uint16` element.

Optionally the `CppClassImplementationDataType` of `category VECTOR` may aggregate a second `templateArgument` that defines the used `Allocator` with the `allocator` reference. The type of the `Allocator` is the same as the data type the vector consists of.

If an `Allocator` is referenced then the attribute `arraySize` in the `CppClassImplementationDataType` of `category VECTOR` can be used to define the maximal size of the vector.

[SWS_CM_00407]{DRAFT} `StdCppImplementationDataType` of `category VECTOR with one dimension defined without an Allocator` [For each `StdCppImplementationDataType` of `category VECTOR` having only one dimension, there shall exist the corresponding type declaration as:

```
using <name> = ara::core::Vector<<element>>;
```

where:

`<name>` is the `CppClassImplementationDataType` symbol of the `Vector Cpp Implementation Data Type`.

`<element>` is the vector element specification. It is defined by the `templateArgument` that refers to a `CppClassImplementationDataType` with the `templateType` reference. The referenced `CppClassImplementationDataType` itself can be one of the data types allowed for the Adaptive Platform.

- If the `CppClassTemplateArgument` is marked with `inplace = false` the `shortName` of the referenced `CppClassImplementationDataType` is used and the declaration of the referenced `CppClassImplementationDataType` is defined **outside** of the vector.
- If the `CppClassTemplateArgument` is marked with `inplace = true` an unnamed `CppClassImplementationDataType` is defined as value type of the vector and the `shortName` of the referenced `CppClassImplementationDataType` is ignored.

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127*)

In case that a `StdCppImplementationDataType` with `category VECTOR` and the `shortName MyVector` has a `CppClassTemplateArgument` that points with the `templateType` reference to a `StdCppImplementationDataType` with `category VECTOR` and the `CppClassTemplateArgument` is marked with `inplace = true` this will result in the following code:

```
1 using MyVector = ara::core::Vector<ara::core::Vector<uint16_t>>;
```

If the `CppClassTemplateArgument` is marked with `inplace = false` this will result in the following code:

```
1 using MyVector = ara::core::Vector<MyInsideVector>;  
2 using MyInsideVector = ara::core::Vector<uint16_t>;
```

[SWS_CM_00503]{DRAFT} `StdCppImplementationDataType` of category **VECTOR** with one dimension defined with an `Allocator` [For each `Vector Cpp Implementation Data Type` having only one dimension and a defined `Allocator` without a defined `arraySize`, there shall exist the corresponding type declaration as:

```
using <name> = ara::core::Vector<<element>,<allocator<element>>>.
```

If an `arraySize` is defined, the corresponding type declaration shall exist as:

```
using <name> = ara::core::Vector<<element>,<allocator<element>,<maxSize>>>;
```

where:

`<name>` is the `CppClass Implementation Data Type symbol` of the `Vector Cpp Implementation Data Type`,

`<element>` is the vector element specification. It is defined by the `templateArgument` that refers to a `CppClassImplementationDataType` with the `templateType` reference. The referenced `CppClassImplementationDataType` itself can be one of the data types allowed for the Adaptive Platform.

- If the `CppClassTemplateArgument` is marked with `inplace = false` the `shortName` of the referenced `CppClassImplementationDataType` is used and the declaration of the referenced `CppClassImplementationDataType` is defined **outside** of the vector.
- If the `CppClassTemplateArgument` is marked with `inplace = true` an unnamed `CppClassImplementationDataType` is defined as value type of the vector and the `shortName` of the referenced `CppClassImplementationDataType` is ignored.

`<allocator>` is the `<allocator namespace>::<allocator shortname>` of the defined `Allocator` that is referenced by a `CppClassTemplateArgument` of `Vector Cpp Implementation Data Type` with the `allocator` reference,

`<maxSize>` is the defined `arraySize` of the `StdCppClassImplementationDataType` of category **VECTOR**.

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127*)

A *multidimensional* `CppClassImplementationDataType` of category **VECTOR** contains nested `CppClassImplementationDataTypes` of category **VECTOR**. This means, that the `CppClassImplementationDataType` of category **VECTOR** will refer to a `CppClassImplementationDataType` of category **VECTOR** via the aggregated `templateArgument`. Such a definition describes a *two-dimensional* `Vector Cpp Implementation Data Type`; consequently a type with more dimensions is described by just

nesting more [CppImplementationDataType](#)s of category VECTOR. The innermost [CppImplementationDataType](#) of category VECTOR has the reference to the type of elements that are contained in the vector.

[SWS_CM_00408]{DRAFT} Vector Data Type with more than one dimension [For each [Vector Cpp Implementation Data Type](#) having more than one dimension, there shall exist the corresponding type declaration according to [\[SWS_CM_00407\]](#) or [\[SWS_CM_00503\]](#) as base where <element> has a nested vector for each additional dimension. The total number of dimensions is equal to the number of nested [CppClassImplementationDataTypes](#) with [category](#) VECTOR plus one for the top level [Vector Cpp Implementation Data Type](#). The vector element itself is specified by the innermost [CppClassImplementationDataType](#) with [category](#) different from VECTOR.] ([RS_CM_00211](#), [RS_CM_00003](#))

For a *two-dimensional* [Vector Cpp Implementation Data Type](#), as it is given as example for the definition of a *Rectangular Vector Data Type* in [5], the corresponding type declaration would look like this:

```
1 using DynamicdataArrayImplRectangular = ara::core::Vector<ara::core::Vector<  
  uint16_t>>;
```

[SWS_CM_00452]{DRAFT} Usage of attribute [arraySize](#) of an [CppClassImplementationDataType](#) with [category](#) VECTOR [The size of an [CppClassImplementationDataType](#) of [category](#) VECTOR that is specified in [CppClassImplementationDataType.arraySize](#) will only be taken into account when the respective [CppClassImplementationDataType](#) defines an [Allocator](#) as defined in [\[SWS_CM_00503\]](#).] ([RS_CM_00211](#), [RS_CM_00003](#))

[SWS_CM_00450]{DRAFT} Define the maximum size of allocated vector memory [The maximum size of usable memory for an [CppClassImplementationDataType](#) of [category](#) VECTOR can be limited using an [Allocator](#) as [CppClassTemplateArgument](#) as defined in [\[SWS_CM_00503\]](#).] ([RS_CM_00211](#), [RS_CM_00003](#))

For more details how to model [Vector Cpp Implementation Data Type](#), see the chapter *Vector Data Type* of AUTOSAR Manifest Specification document [5].

With the usage of [CustomCppClassImplementationDataType](#) it is possible to specify a data type definition that is taken as the basis for a C++ language binding to a custom implementation that is declared in the configured [headerFile](#). In case of a [CustomCppClassImplementationDataType](#) the model defines the following:

- Class-Name of the custom implementation ([CustomCppClassImplementationDataType.shortName](#))
- Namespace of the custom implementation ([CustomCppClassImplementationDataType.namespace](#))
- Header File that contains the custom class declaration ([CustomCppClassImplementationDataType.headerFile](#)).

[SWS_CM_00507]{DRAFT} **CustomCppImplementationDataType of category VECTOR** If a **CustomCppImplementationDataType** of category **VECTOR** is used that contains a single **templateArgument** that refers to a **CppType** with the **templateType** reference the following type declaration shall be available in the included **headerFile**:

```
1 <ClassName><<element>>;
```

For each **CustomCppImplementationDataType** of category **VECTOR** and a defined **Allocator** without a defined **arraySize**, there shall exist the corresponding type declaration as:

```
<ClassName><<element>,<allocator<element>>>
```

If an **arraySize** is defined, the corresponding type declaration shall exist as:

```
<ClassName><<element>,<allocator<element>,<maxSize>>>
```

where:

<ClassName> is the **CppType** symbol of the **CustomCppMethodImplementationDataType** of category **VECTOR**. Please note that the namespace that is defined with an ordered list of defined **symbol** is already handled by **[SWS_CM_10375]**,

<element> is the vector element specification. It is defined by the **templateArgument** that refers to the vector element with the **templateType** reference,

<allocator> is the **<allocator namespace>::<allocator shortname>** of the defined **Allocator** that is referenced by a **CppType** of **Vector Cpp Implementation Data Type** with the **allocator** reference,

<size> is the defined **arraySize**.

] (RS_AP_00113, RS_AP_00114)

Please note that multidimensional **CustomCppMethodImplementationDataTypes** of category **VECTOR** are handled in the same way as **StdCppMethodImplementationDataTypes** of category **VECTOR**. **[SWS_CM_00408]** is also valid for **CustomCppMethodImplementationDataTypes** of category **VECTOR**.

8.1.2.5.9 Associative Map Implementation Data Type

The Communication Management declares C++ types for all **Associative Map Cpp Implementation Data Types** defined in the **ServiceInterface**. In AUTOSAR Adaptive Platform, the C++ binding of an **Associative Map Cpp Implementation Data Type** could either be implemented as an **ara::core::Map** or as a custom map.

[SWS_CM_00409]{DRAFT} **StdCppImplementationDataType with category ASSOCIATIVE_MAP defined without an Allocator** [For each StdCppImplementationDataType with category ASSOCIATIVE_MAP , there shall exist the corresponding type declaration as:

```
using <name> = ara::core::Map<<key>, <value>>;
```

where:

<**name**> is the Cpp Implementation Data Type symbol of the Associative Map Cpp Implementation Data Type,

<**key**> is the map key type specification. It is defined by the first CppTemplateArgument which is aggregated by the Associative Map Cpp Implementation Data Type and points to a CppImplementationDataType with the templateType reference. The referenced CppImplementationDataType itself can be one of the data types allowed for the Adaptive Platform as long as the requirements on the key data type imposed by the ara::core::Map implementation (namely the applicability of std::less<key>) are met.

- If the CppTemplateArgument is marked with `inplace = false` the shortName of the referenced CppImplementationDataType is used and the declaration of the referenced CppImplementationDataType is defined **outside** of the map.
- If the CppTemplateArgument is marked with `inplace = true` an unnamed CppImplementationDataType is defined as key type and the shortName of the referenced CppImplementationDataType is ignored.

<**value**> is the mapped value type specification. It is defined by the second CppTemplateArgument which is aggregated by the Associative Map Cpp Implementation Data Type and points to a CppImplementationDataType with the templateType reference. The CppImplementationDataType itself can be one of the data types allowed for the Adaptive Platform.

- If the CppTemplateArgument is marked with `inplace = false` the shortName of the referenced CppImplementationDataType is used and the declaration of the referenced CppImplementationDataType is defined **outside** of the map.
- If the CppTemplateArgument is marked with `inplace = true` an unnamed CppImplementationDataType is defined as value type and the shortName of the referenced CppImplementationDataType is ignored.

] (RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127)

For an Associative Map Cpp Implementation Data Type as it is given as example in chapter *Associative Map Data Type* of [5], the corresponding type declaration would look like this:

```
1 using MyMap = ara::core::Map<uint16_t, uint8_t>;
```

[SWS_CM_00505]{DRAFT} **StdCppImplementationDataType with category ASSOCIATIVE_MAP defined with an Allocator** [For each StdCppImplementationDataType with category ASSOCIATIVE_MAP with a defined Allocator, there shall exist the corresponding type declaration as:

```
using <name> =
ara::core::Map<<key>, <value>, std::less<<key>>, <allocator>>;
```

where:

<**name**> is the Cpp Implementation Data Type symbol of the Associative Map Cpp Implementation Data Type,

<**key**> is the map key type specification. It is defined by the first CppTemplateArgument which is aggregated by the Associative Map Cpp Implementation Data Type and points to a CppImplementationDataType with the templateType reference. The referenced CppImplementationDataType itself can be one of the data types allowed for the Adaptive Platform as long as the requirements on the key data type imposed by the ara::core::Map implementation (namely the applicability of std::less<key>) are met.

- If the CppTemplateArgument is marked with `inplace = false` the shortName of the referenced CppImplementationDataType is used and the declaration of the referenced CppImplementationDataType is defined outside of the map.
- If the CppTemplateArgument is marked with `inplace = true` an unnamed CppImplementationDataType is defined as key type and the shortName of the referenced CppImplementationDataType is ignored.

<**value**> is the mapped value type specification. It is defined by the second CppTemplateArgument which is aggregated by the Associative Map Cpp Implementation Data Type and points to a CppImplementationDataType with the templateType reference. The CppImplementationDataType itself can be one of the data types allowed for the Adaptive Platform.

- If the CppTemplateArgument is marked with `inplace = false` the shortName of the referenced CppImplementationDataType is used and the declaration of the referenced CppImplementationDataType is defined outside of the map.
- If the CppTemplateArgument is marked with `inplace = true` an unnamed CppImplementationDataType is defined as value type and the shortName of the referenced CppImplementationDataType is ignored.

<**allocator**> is the defined Allocator that is referenced by the third CppTemplateArgument of Associative Map Cpp Implementation Data Type with the allocator reference.

] (RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114, RS_AP_00127)

With the usage of `CustomCppImplementationDataType` it is possible to specify a data type definition that is taken as the basis for a C++ language binding to a custom implementation that is declared in the configured `headerFile`. In case of a `CustomCppMethodImplementationDataType` the model defines the following:

- Class-Name of the custom implementation (`CustomCppMethodImplementationDataType.shortName`)
- Namespace of the custom implementation (`CustomCppMethodImplementationDataType.namespace`)
- Header File that contains the custom class declaration (`CustomCppMethodImplementationDataType.headerFile`).

[SWS_CM_00506]{DRAFT} `CustomCppMethodImplementationDataType` of category ASSOCIATIVE_MAP [If a `CustomCppMethodImplementationDataType` of category ASSOCIATIVE_MAP is used that contains two `templateArguments`s that both refer to a `CppMethodImplementationDataType` with the `templateType` reference the following type declaration shall be available in the included `headerFile`:

```
1 <ClassName><<key>, <value>>;
```

For each `CustomCppMethodImplementationDataType` of category ASSOCIATIVE_MAP and a defined `Allocator` the following type declaration shall be available in the included `headerFile`:

```
<ClassName><<key>, <value>, <compare>, <allocator>>
```

where:

`<ClassName>` is the `CppMethod Implementation Data Type symbol` of the `CustomCppMethodImplementationDataType` of category ASSOCIATIVE_MAP. Please note that the `namespace` that is defined with an ordered list of defined `symbol` is already handled by [SWS_CM_10375],

`<key>` is the map key type specification. It is defined by the first `CppTypeArgument` which is aggregated by the `Associative Map Cpp Implementation Data Type` and points to a `CppMethodImplementationDataType` with the `templateType` reference. The referenced `CppMethodImplementationDataType` itself can be one of the data types allowed for the Adaptive Platform,

`<value>` is the mapped value type specification. It is defined by the second `CppTypeArgument` which is aggregated by the `Associative Map Cpp Implementation Data Type` and points to a `CppMethodImplementationDataType` with the `templateType` reference. The `CppMethodImplementationDataType` itself can be one of the data types allowed for the Adaptive Platform,

`<compare>` is the comparison function used to sort the keys.

`<allocator>` is the defined `Allocator` that is referenced by the third `CppTypeArgument` of `Associative Map Cpp Implementation Data Type` with the `allocator` reference.

](*RS_AP_00113, RS_AP_00114*)

For more details how to model [Associative Map Cpp Implementation Data Type](#), see the chapter *Map Data Type* of AUTOSAR Manifest Specification document [5].

8.1.2.5.10 Redefinition of Implementation Data Type

[SWS_CM_00410]{DRAFT} Data Type redefinition [For each [Redefinition Cpp Implementation Data Type](#) which is typed by an [StdCppImplementation-DataType](#), there shall exist the corresponding type declaration as:

using <name> = <type>;

where:

<name> is the [Cpp Implementation Data Type symbol](#) of the [Redefinition Cpp Implementation Data Type](#),

<type> is the [Cpp Implementation Data Type symbol](#) of the referred [StdCppImplementationDataType](#).

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114*)

Please note that the usage of the [category TYPE_REFERENCE](#) is restricted to [StdCppImplementationDataTypes](#) according to [constr_1578] defined in [5] for simplification reasons.

8.1.2.5.11 Enumeration Data Types

An Enumeration is not a plain primitive data type, but a structural description defined with a set of custom identifiers known as *enumerators* representing the possible values. In C++, an Enumeration is a first-class object and can take any of these enumerators as a value.

It is recommended that the underlying type of the enumeration should be explicitly defined to achieve both type safety and a fixed, well-defined size. Additionally, declaring enumerations as scoped enumeration classes avoids the need of unique enumerator names.

Therefore enumerations being both typed and scoped are used instead of classic C++ enumerations; the underlying type must be provided by the input configuration by defining an [Enumeration Data Type](#).

[SWS_CM_00424]{DRAFT} Enumeration Data Type [For each [Enumeration Data Type](#) referenced by the [ServiceInterface](#), there shall exist the corresponding type declaration as:

```
enum class <name> : <type> {  
    <enumerator-list>  
};
```

where:

<**name**> is the Cpp Implementation Data Type symbol of the Redefinition Cpp Implementation Data Type that boils down to a Primitive Cpp Implementation Data Type.

<**type**> is the Primitive Cpp Implementation Data Type that is referenced by the Redefinition Cpp Implementation Data Type.

<**enumerator-list**> are the enumerators as defined by [SWS_CM_00425].

] (RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114)

The enumerator names base on the CompuScale code symbolic name as defined in [TPS_SWCT_01569] of the AUTOSAR Software Component Template [29].

[SWS_CM_00425]{DRAFT} **Definition of enumerators** [For each CompuScale with point range (i.e., lowerLimit equals upperLimit and both lowerLimit.intervalType and upperLimit.intervalType are either missing or set to CLOSED) in the Enumeration Data Type, there shall exist the corresponding enumeration nested in the declaration defined by [SWS_CM_00425] as:

```
<enumeratorLiteral> = <initializer><suffix>,
```

where:

<**enumeratorLiteral**> is the name of the enumerator according to the following rule (lower values indicate higher priority):

1. the C++ compliant identifier specified by the symbol attribute of CompuScale if this attribute is available and not empty,
2. the string specified by the value of vt element of the CompuConst of the CompuScale if the value is a valid C++ identifier,
3. the string specified by the value of shortLabel attribute of CompuScale if the attribute is available and not empty.

<**initializer**> is the CompuScale's point range used as enumerator initializer,

<**suffix**> shall be "U" if <**type**> of [SWS_CM_00424] is an unsigned data type (i.e. if the Redefinition Cpp Implementation Data Type boils down to a Primitive Cpp Implementation Data Type where the Cpp Implementation Data Type symbol equals uint8_t, uint16_t, uint32_t or uint64_t. <**suffix**> shall empty if it is a signed data type (i.e. if the Redefinition Cpp Implementation Data Type boils down to a Primitive Cpp Implementation Data Type where the Cpp Implementation Data Type symbol equals int8_t, int16_t, int32_t or int64_t.

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114*)

[SWS_CM_10376]{DRAFT} Skip CompuScales with non-point range [Any CompuScale with non-point range shall be simply skipped, i.e., no enumeration according to [SWS_CM_00425] shall be generated for those CompuScales.](*RS_CM_00211, RS_CM_00003*)

[SWS_CM_00426]{DRAFT} Reject incomplete Enumeration Data Types [If the input configuration contains an Enumeration Data Type and the name of an enumerator can not be determined according to [SWS_CM_00425], the ARA generator shall reject this input as an invalid configuration.](*RS_CM_00211, RS_CM_00003*)

8.1.2.5.12 Scale Linear And Texttable Data Types

A Scale Linear And Texttable Data Type is not a plain primitive data type, but a structural description defined with an Enumeration Data Type. The Scale Linear And Texttable Data Type can hold the values of the enumerators and also the values of the underlying type of the Enumeration Data Type it was defined with.

The Communication Management declares C++ types for all Scale Linear And Texttable Data Types defined in the ServiceInterface. In AUTOSAR Adaptive Platform, the C++ binding of a Scale Linear And Texttable Data Type is always implemented by an `ara::com::ScaleLinearAndTexttable`.

[SWS_CM_10409]{DRAFT} Scale Linear And Texttable type definition [For each Scale Linear And Texttable Data Type there shall exist the corresponding type declaration as:

```
using <name> = ScaleLinearAndTexttable<enum_type>;
```

where:

<name> is the Cpp Implementation Data Type symbol of the Scale Linear And Texttable Data Type,

<enum_type> is the generated Enumeration Data Type used to specify the Scale Linear And Texttable Data Type.

](*RS_CM_00211, RS_CM_00003, RS_AP_00113, RS_AP_00114*) For the specification of Enumeration Data Type see section 8.1.2.5.11).

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::E2EEError (see chapter [8.1.2.7](#))
3. Application Errors: These errors specific to a certain service interface call. They are defined as [ApApplicationError](#) in the meta-model.

Errors can also occur in a call to a [RawDataStreamInterface](#) 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:	ara::com::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 committed in Subscribe().
	kNetworkBindingFailure= 3	Local failure has been detected by the network binding.
Header file:	#include "ara/com/com_error_domain.h"	
Description:	The @ARTechTerm{ComErrc} enumeration defines the error codes for the ComErrorDomain. .	

] ([RS_AP_00130](#))

[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_00113, RS_AP_00114, RS_AP_00119, RS_AP_00127*)

[SWS_CM_11268]{DRAFT} **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_12367]{DRAFT} [

Kind:	enumeration	
Symbol:	ara::com::raw::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.
Header file:	#include "ara/com/raw/raw_error_domain.h"	
Description:	The @ARTechTerm{RawErrc} enumeration defines the error codes for the RawErrorDomain. .	

](*RS_AP_00130*)

8.1.2.7 E2E Related Data Types

Some data types are used only in context of e2e-protected communication of events.

[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).
- **kNotAvailable:** No value has been received yet (e.g. during initialization). This is used as the initialization value for the buffer.
- **kNoNewData:** No new data is available (assuming a sample has already been received since the initialization).
- **kCheckDisabled:** No E2E check status available (no E2E protection is configured).

```
1 enum class ProfileCheckStatus : uint8_t
2 {
3     kOk,
4     kRepeated,
5     kWrongSequence,
6     kError,
7     kNotAvailable,
8     kNoNewData,
9     kCheckDisabled
10 };
```

]([RS_E2E_08534](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00115](#), [RS_AP_00119](#))

The E2E state machine `SMState` is determined by checking a history of `ProfileCheckStatuses`. 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_state_machine::E2EState` [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 (no statemachine is configured).

```
1 enum class SMState : uint8_t
```

```

2 {
3     kValid,
4     kNoData,
5     kInit,
6     kInvalid,
7     kStateMDisabled
8 };

```

]([RS_E2E_08534](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00115](#), [RS_AP_00119](#))

The Result is a class providing ProfileCheckStatus and SMState.

[SWS_CM_10474]{DRAFT} [

Kind:	enumeration	
Symbol:	ara::com::e2e::E2EError	
Scope:	namespace ara::com::e2e	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	enum class E2EError : ara::core::ErrorDomain::CodeType {...};	
Values:	repeated= 1	Data has a repeated counter.
	wrong_sequence_error= 2 error = 3	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta. 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.
	not_available= 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.
	no_new_data= 5	No new data is available.
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Description:	The @ARTechTerm{E2EError} enumeration defines the error codes for the E2EErrorDomain. .	

]([RS_AP_00130](#))

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.]

```
1 class UpperCamelCase(<ServiceInterface.shortName>) Skeleton {  
2 ...  
3 };
```

](RS_CM_00101, RS_AP_00113, 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].]

```
1 class UpperCamelCase(<VariableDataPrototype.shortName>) {  
2 ...  
3 };
```

](RS_CM_00201, RS_AP_00113, 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].]

```
1 class UpperCamelCase(<Field.shortName>) {  
2 ...  
3 };
```

](RS_CM_00219, RS_AP_00113, 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.]

```
1 class UpperCamelCase(<ServiceInterface.shortName>) Proxy {  
2 ...  
3 };
```

](RS_CM_00102, RS_AP_00113, 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].

```
1 class UpperCamelCase(<VariableDataPrototype.shortName>) {  
2 ...  
3 };
```

](RS_CM_00103, RS_AP_00113, 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].

```
1 class UpperCamelCase(<ClientServerOperation.shortName> {  
2 ...  
3 };
```

](RS_CM_00212, RS_CM_00213, RS_AP_00113, 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].

```
1 class UpperCamelCase(<Field.shortName>) {  
2 ...  
3 };
```

](RS_CM_00216, RS_AP_00113, RS_AP_00114)

The following sub-chapters describe the content of the previously defined classes.

8.1.3.1 Object Creation via Construction Token

The construction token approach enables exception-less error reporting for object construction. Since service skeletons and service proxies can be created using a `ConstructionToken`, 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 8.1.3.10, respectively.

[SWS_CM_10433]{DRAFT} Declaration of Construction Token [The construction token shall be declared within the namespace of the class to be created `ClassToBeCreated::ConstructionToken`. The token must hold all members which are necessary to instantiate a valid object of `ClassToBeCreated`. The `ConstructionToken` shall implement move-only semantics.

```
ConstructionToken(ConstructionToken &&);  
ConstructionToken& operator=(ConstructionToken &&);
```

```
ConstructionToken(const ConstructionToken&) = delete;
ConstructionToken& operator=(const ConstructionToken&) = delete;
](RS_CM_00101, RS_AP_00113, RS_AP_00114)
```

[SWS_CM_10434]{DRAFT} Creation of a Construction Token [The `ClassToBeCreated` shall provide a static member function `Preconstruct` returning the construction token embedded in an `ara::core::Result`. This function performs all operations for constructing an object of `ClassToBeCreated` which may fail or result in an error, e.g. parameter checks and resource allocation. If an error occurs during these operations, the error is returned as `ara::core::ErrorCode`. A non-throwing constructor of `ClassToBeCreated` shall take the `ConstructionToken` as r-value reference.]

```
static ara::core::Result<ConstructionToken>
    Preconstruct /* construction arguments */;
ClassToBeCreated (ConstructionToken&&) noexcept;
](RS_CM_00101, RS_AP_00113, 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 [7.7.1](#).

[SWS_CM_00101]{DRAFT} 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.]

```
void OfferService();
](RS_CM_00101, RS_AP_00113, RS_AP_00114, RS_AP_00120)
```

[SWS_CM_00111]{DRAFT} 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_00113, 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 [7.7.2](#).

[SWS_CM_00130]{DRAFT} 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_00113, RS_AP_00114, RS_AP_00121)
```

[SWS_CM_10435]{DRAFT} Exception-less creation of service skeleton using Instance ID [The Communication Management shall provide a non-throwing constructor for each specific ServiceSkeleton class according to [SWS_CM_10433] and [SWS_CM_10434]. A Preconstruct function 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.

```
ServiceSkeleton::ServiceSkeleton(ConstructionToken&&) noexcept;  
  
static ara::core::Result<ConstructionToken>  
    ServiceSkeleton::Preconstruct(  
        ara::com::InstanceIdentifier instanceID,  
        ara::com::MethodCallProcessingMode mode =  
            ara::com::MethodCallProcessingMode::kEvent  
) ;  
)  
(RS_CM_00101, RS_AP_00113, RS_AP_00114, RS_AP_00115, RS_AP_00121,  
RS_AP_00139, RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00139)
```

[SWS_CM_00152]{DRAFT} 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_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::core::InstanceSpecifier instanceSpec,  
    ara::com::MethodCallProcessingMode mode =  
        ara::com::MethodCallProcessingMode::kEvent  
) ;  
  
](RS\_CM\_00101, RS\_AP\_00113, RS\_AP\_00114, RS\_AP\_00115, RS\_AP\_00121,  
RS\_AP\_00127, RS\_AP\_00137)
```

[SWS_CM_10436]{DRAFT} Exception-less creation of service skeleton using Instance Spec [The Communication Management shall provide a constructor for each specific ServiceSkeleton class according to [[SWS_CM_10433](#)] and [[SWS_CM_10434](#)]. A Preconstruct function 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_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(ConstructionToken&&) noexcept;  
static ara::core::Result<ConstructionToken> Preconstruct(  
    ara::core::InstanceSpecifier instanceSpec,  
    ara::com::MethodCallProcessingMode mode =  
        ara::com::MethodCallProcessingMode::kEvent  
) ;  
  
](RS\_CM\_00101, RS\_AP\_00113, 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)
```

[SWS_CM_00153]{DRAFT} 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_00113, RS_AP_00114, RS_AP_00115, RS_AP_00121)
```

[SWS_CM_10437]{DRAFT} Exception-less creation of service skeleton using Instance ID Container [The Communication Management shall provide a constructor for each specific ServiceSkeleton class according to [SWS_CM_10433] and [SWS_CM_10434]. A Preconstruct function 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.

```
ServiceSkeleton(ConstructionToken&&) noexcept;  
static ara::core::Result<ConstructionToken> Preconstruct(  
    ara::com::InstanceIdentifierContainer instanceIDs,  
    ara::com::MethodCallProcessingMode mode =  
        ara::com::MethodCallProcessingMode::kEvent  
) ;
```

```
](RS_CM_00101, RS_AP_00113, RS_AP_00114, RS_AP_00115, RS_AP_00121,  
RS_AP_00139, RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00139)
```

[SWS_CM_00134]{DRAFT} 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_00113, RS_AP_00114)
```

[SWS_CM_00135]{DRAFT} 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_00113, RS_AP_00114)
```

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.]

```
void Event::Send(const SampleType &data);  
](RS_CM_00201, RS_AP_00113, 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.]

```
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.]
(RS_CM_00201, RS_AP_00113, 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.]

```
ara::com::SampleAllocateePtr <SampleType> Event::Allocate();
```

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.]
(RS_CM_00201, RS_AP_00113, 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.

8.1.3.5 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 and optional return values shall be provided inside the ServiceSkeleton class.

The method shall return an ara::core::Future object wrapping the output parameters and return values 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;  
    ...  
    TypeResult result;  
};  
  
virtual ara::core::Future <Method1Output> Method1(  
    TypeInputParameter1 input1,  
    TypeInputParameter2 input2,  
    ...  
) = 0;  
  
](RS_CM_00211, RS_AP_00113, 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 methods 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_00113](#), [RS_AP_00114](#))

8.1.3.6 Processing of service methods

For the functional description of the processing of service methods API, see chapter [7.7.3](#).

[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_00113](#), [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. Additionally, the returned `ara::core::Future` object allows to register a callback function which is invoked when the next pending execution of a method request is finished.

```
ara::core::Future<bool> ProcessNextMethodCall();
```

]([RS_CM_00211](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00120](#), [RS_AP_00138](#),
[RS_AP_00128](#), [RS_AP_00127](#), [RS_AP_00138](#))

[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_00113](#), [RS_AP_00114](#), [RS_AP_00119](#),
[RS_AP_00127](#))

8.1.3.7 Registering get handlers for fields

For the functional description of the registering get handlers for fields API, see chapter 7.7.4.

[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.

```
void RegisterGetHandler(  
    std::function<ara::core::Future<FieldType>(  
        )> getHandler);
```

]([RS_CM_00218](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00120](#), [RS_AP_00138](#),
[RS_AP_00128](#), [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_00113](#), [RS_AP_00114](#))

8.1.3.8 Registering set handlers for fields

For the functional description of the registering set handlers for fields API, see chapter 7.7.5.

[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.

```
void RegisterSetHandler(  
    std::function<ara::core::Future<FieldType>(  
        const FieldType& value)> setHandler);
```

]([RS_CM_00218](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00120](#), [RS_AP_00138](#),
[RS_AP_00128](#), [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_00113](#), [RS_AP_00114](#))

[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:

```
void Field::Update(const FieldType &value);  
](RS\_CM\_00218, RS\_AP\_00113, RS\_AP\_00114, RS\_AP\_00120, RS\_AP\_00121)
```

8.1.3.9 Find service

For the functional description of the find service API, see chapter [7.7.6](#).

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.

As result a container containing handles for all matching service instances is returned. There is one `FindService` method for using a specified `InstanceIdentifier`.

```
static ara::com::ServiceHandleContainer<<ProxyClassName>::HandleType>
    FindService(ara::com::InstanceIdentifier instance);
```

where `<ProxyClassName>` is the name of the `ServiceProxy` class as defined in [\[SWS_CM_00004\].](#)([RS_CM_00102](#), [RS_AP_00113](#), [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.

As result a container containing handles for all matching service instances is returned.

```
static ara::com::ServiceHandleContainer<<ProxyClassName>::HandleType>
    FindService(ara::core::InstanceSpecifier instance);
```

where `<ProxyClassName>` is the name of the `ServiceProxy` class as defined in [\[SWS_CM_00004\].](#)([RS_CM_00102](#), [RS_AP_00113](#), [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`,
- [\[SWS_CM_00350\]](#) for `InstanceSpecifier`.

[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. As result a FindServiceHandle for this search/find request is returned, which 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::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].] ([RS_CM_00102](#), [RS_AP_00113](#), [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.

[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. As result a FindServiceHandle for this search/find request is returned, which is needed to stop the service availability monitoring and related firing of the given handler.

```
static 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].] ([RS_CM_00102](#), [RS_AP_00113](#), [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_CM_00350] for InstanceSpecifier.

[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_00113, RS_AP_00113, RS_AP_00114, RS_AP_00120,  
RS_AP_00121)
```

See [SWS_CM_00303] for the type definition of FindServiceHandle.

8.1.3.10 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_00113, RS_AP_00114, RS_AP_00121)
```

[SWS_CM_10438]{DRAFT} Exception-less creation of service proxy [The Communication Management shall provide a non-throwing constructor for each specific ServiceProxy class according to [SWS_CM_10433] and [SWS_CM_10434]. A Preconstruct function shall take a handle returned by any FindService method of the ServiceProxy class.]

```
explicit ServiceProxy::ServiceProxy(ConstructionToken&&) noexcept;  
static ara::core::Result<ConstructionToken>  
    ServiceProxy::Preconstruct(  
        const HandleType &handle);  
](RS_CM_00102, RS_AP_00113, RS_AP_00114, RS_AP_00121, RS_AP_00139,  
RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00139)
```

[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_00113, RS_AP_00114, RS_AP_00119)
```

See [SWS_CM_00312] for the type definition of HandleType.

[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\_00113, RS\_AP\_00114)
```

[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\_00113, RS\_AP\_00114)
```

8.1.3.11 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_00113](#), [RS_AP_00114](#), [RS_AP_00127](#))

8.1.3.12 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.]

```
void Event::Subscribe(  
    size_t maxSampleCount  
) ;  
](RS\_CM\_00103, RS\_AP\_00113, 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, otherwise error handling according to [SWS_CORE_00001] shall be performed.]
([RS_CM_00103](#), [RS_AP_00113](#), [RS_AP_00114](#))

[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();  
]  
(RS\_CM\_00104, RS\_AP\_00113, 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\_00113, 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.

```
1 void SetSubscriptionStateChangeHandler(ara::com::  
SubscriptionStateChangeHandler handler);  
  
]  
(RS\_CM\_00106, RS\_AP\_00113, 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.

```
1 void UnsetSubscriptionStateChangeHandler();  
  
]  
(RS\_CM\_00106, RS\_AP\_00113, 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_00113](#),
[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_00113](#),
[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_00113](#), [RS_AP_00114](#))

8.1.3.13 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]{DRAFT} 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<size_t> GetNewSamples(
    F& f,
    size_t maxNumberOfSamples = std::numeric_limits<size_t>::max());
```

]([RS_CM_00202](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00120](#), [RS_AP_00121](#),
[RS_AP_00139](#), [RS_AP_00128](#), [RS_AP_00127](#), [RS_AP_00139](#))

[SWS_CM_00702]{DRAFT} 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_00113](#), [RS_AP_00114](#))

[SWS_CM_00703]{DRAFT} 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.
- deserialize the data, if needed.
- place the 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 `SamplePtrs` 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.

]([RS_CM_00202](#), [RS_AP_00113](#), [RS_AP_00114](#))

[SWS_CM_00704]{DRAFT} Return Value [The returned `ara::core::Result` either contains a

- `size_t` indicating the number of data samples passed to `f` in the context of the call.

or a

- `ara::core::ErrorCode` with value `kMaxSamplesReached` indicating, that applications `SamplePtrs` count has been reached.

]([RS_CM_00202](#), [RS_AP_00113](#), [RS_AP_00114](#), [RS_AP_00119](#), [RS_AP_00127](#))

[SWS_CM_00714]{DRAFT} Reentrancy [GetNewSamples shall be re-entrant for different `ServiceProxy` class instances. When called concurrently on the same `ServiceProxy` class instance, the behavior is undefined.]([RS_CM_00202](#), [RS_AP_00113](#), [RS_AP_00114](#))

For the E2E-protected events, after updating the event cache via the `GetNewSamples` method, and before accessing the `SamplePtrs`, the current `Result` needs to be retrieved by calling the `GetResult` method.

[SWS_CM_90424]{DRAFT} Provide E2E Result [Inside the specific E2E-protected Events belonging to the specific ServiceProxy class, the method GetResult shall be provided.

```
const ara::com::e2e::Result GetResult() const;  
](RS\_E2E\_08534, RS\_AP\_00113, RS\_AP\_00114, RS\_AP\_00115)
```

[SWS_CM_00705]{DRAFT} 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.

```
size_t GetFreeSampleCount() const noexcept;  
](RS\_CM\_00202, RS\_AP\_00113, RS\_AP\_00114, RS\_AP\_00120, RS\_AP\_00139,  
RS\_AP\_00128, RS\_AP\_00132, RS\_AP\_00127, RS\_AP\_00139)
```

[SWS_CM_00706]{DRAFT} 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_00113](#), [RS_AP_00114](#), [RS_AP_00119](#), [RS_AP_00139](#), [RS_AP_00128](#), [RS_AP_00127](#))

[SWS_CM_00707]{DRAFT} 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 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\_00113, RS\_AP\_00114)
```

8.1.3.14 Receive event by getting triggered

For the functional description of the receive event by getting triggered API, see chapter [7.7.7.2](#).

[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.

```
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.] (RS_CM_00203, RS_AP_00113, 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.

```
void Event::UnsetReceiveHandler()  
](RS_CM_00203, RS_AP_00113, RS_AP_00114, RS_AP_00120)
```

8.1.3.15 Call a service method

For the functional description of the call a service method API, see chapter 7.7.8.

[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`) and optional return values 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;  
        ...  
        TypeResult result; // return value (optional)  
    };  
  
    ara::core::Future<Output> operator()(  
        TypeInputParameter1 input1,  
        TypeInputParameter2 input2,  
        ...  
    );
```

```
) ;  
};  
](RS\_CM\_00212, RS\_CM\_00213, RS\_AP\_00113, 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_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_00113](#), [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_00113](#), [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_00113](#), [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 7.5.1) 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_00113](#), [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_00113](#), [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_00113, 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_00113, RS_AP_00114, RS_AP_00120*)

8.1.3.16 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_00113, 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_00113, RS_AP_00114*)

8.1.3.17 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_00113, 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_00113, RS_AP_00114*)

8.1.3.18 Instance Specifier Translation

For the functional description of the Instance Specifier Translation API, see chapter [7.7.10](#).

[SWS_CM_00118]{DRAFT} Method Instance Specifier Translation [The Communication Management shall provide ResolveInstanceIDs method to translate an InstanceSpecifier to a Instance Identifiers list. The size of the list could be 0, 1 or greater than 1 depending on the match.

```
ara::com::InstanceIdentifierContainer  
ara::com::runtime::ResolveInstanceIDs(ara::core::InstanceSpecifier modelName);
```

For the definition of the types used in the ResolveInstanceIDs signature, see:

- [\[SWS_CM_00319\]](#) for InstanceIdentifierContainer,
- [\[SWS_CM_00350\]](#) for InstanceSpecifier.

```
](RS_CM_00207, RS_AP_00113, RS_AP_00114, RS_AP_00115, RS_AP_00120,  
RS_AP_00121, RS_AP_00119, RS_AP_00127, RS_AP_00137)
```

8.1.3.19 Raw Data Stream API

For the functional description of the Raw Data Stream API, see chapter [7.4.2](#).

[SWS_CM_10481]{DRAFT} Class RawDataStream [The Communication Management shall provide a `RawDataStream` class that defines a Raw Data Stream connection. It shall define a constructor and a destructor and four methods for application use.

The class shall be defined in the namespace `ara::com::raw`] ([RS_CM_00410](#), [RS_CM_00411](#))

[SWS_CM_10482]{DRAFT} RawDataStream Constructor [The Communication Management shall provide a constructor that takes an Instance Specifier as parameter. The Instance specifier qualifies the wanted network binding and parameters for the instance.

```
RawDataStream(const ara::core::InstanceSpecifier& instance) noexcept;
```

For the definition of the types used in the constructor signature, see:

- [\[SWS_CM_00350\]](#) for `InstanceSpecifier`,

] ([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

[SWS_CM_10483]{DRAFT} RawDataStream Destructor [The Communication Management shall provide a destructor that deletes the `RawDataStream` instance. If the connection is still open when the destructor is called, it shall be shut down before destroying the `RawDataStream` object.

```
~RawDataStream() noexcept;
```

] ([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

[SWS_CM_10484]{DRAFT} Method Connect [The Communication Management shall provide a `Connect` method to setup a socket connection for the Raw Data Stream defined by the instance. The method shall initialize the socket and establish 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 network endpoint and other parameters are specified in the manifest which is accessed through the `InstanceSpecifier` provided in the constructor.

If TLS security protocol is configured for the socket connection, the TLS/DTLS connection shall be initialized here.

It can optionally take a timeout value as parameter to shorten the configured timeout value in the deployment. The timeout passed as parameter shall only be applied if it is less or equal to the configured one.

It shall return `void` if the connection is successful, or an `ara::core::ErrorCode` from the `ara::com::RawErrorDomain` indicating the error if not successful.

The following errors from the `ara::core::RawErrorDomain` is possible:

- `kConnectionRefused`: The connection was refused by target.

- kAddressNotAvailable: The specified address is not available from the local machine.
- kCommunicationTimeout: The connect operation timed out.
- kStreamAlreadyConnected: The specified connection is already connected.

```
ara::core::Result<void> Connect();  
ara::core::Result<void> Connect(std::chrono::milliseconds timeout);  
](RS_CM_00410, RS_CM_00411, RS_CM_00412)
```

Note: For TLS/DTLS connection with Raw Data Streaming see also chapter [7.6.2.3](#).

[SWS_CM_10485]{DRAFT} Method Shutdown [The Communication Management shall provide a `Shutdown` method to close the socket connection for the Raw Data Stream defined by the instance.

It can optionally take a timeout value as parameter to shorten the configured timeout value in the deployment. The timeout passed as parameter shall only be applied if it is less or equal to the configured one.

It shall return `void` if the connection is successful, or an `ara::core::ErrorCode` from the `ara::com::RawErrorDomain` indicating the error if not successful.

The following errors from the `ara::core::raw::RawErrorDomain` is possible:

- kStreamNotConnected: Trying to shutdown a `RawDataStream` without an established connection.
- kCommunicationTimeout: The operation did not finish until the timeout expired.

```
ara::core::Result<void> Shutdown();  
ara::core::Result<void> Shutdown(std::chrono::milliseconds timeout);  
](RS_CM_00410, RS_CM_00411, RS_CM_00412)
```

[SWS_CM_10486]{DRAFT} Method ReadData [The Communication Management shall provide a `ReadData` method to read a number of bytes from the socket connection for the Raw Data Stream defined by the instance. As input parameter the `ReadData` method expects the requested number of bytes specified in parameter `length`. It can also optionally take a timeout value as parameter to shorten the configured timeout value in the deployment. The timeout passed as parameter shall only be applied if it is less or equal to the configured one.

It shall return a result struct consisting of a pointer to the memory containing the read data, and the actual number of bytes read, or an `ara::core::ErrorCode` from the `ara::com::RawErrorDomain` indicating the error if not successful.

The following errors from the `ara::core::raw::RawErrorDomain` is possible:

- kStreamNotConnected: Trying to shutdown a `RawDataStream` without an established connection.

- kCommunicationTimeout: No data was read until the timeout expired.

```
struct ReadDataResult {  
    ara::com::SamplePtr<uint8_t> data;  
    size_t numberOfBytes;  
};  
  
ara::core::Result<ReadDataResult> ReadData(size_t length);  
ara::core::Result<ReadDataResult> ReadData(size_t length,  
                                         std::chrono::milliseconds timeout);
```

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 `ReadDataResult.data` value.]([RS_CM_00410](#), [RS_CM_00411](#), [RS_CM_00412](#))

[SWS_CM_10487]{DRAFT} Method WriteData [The Communication Management shall provide `WriteData` method to write a number of bytes to the socket connection for the Raw Data Stream defined by the instance. The data is provided in a buffer given in parameter `data`. The number of bytes to write is provided in parameter `length`. It can also optionally take a timeout value as parameter to shorten the configured timeout value in the deployment. The timeout passed as parameter shall only be applied if it is less or equal to the configured one.

For efficiency, the zero-copy semantics of `ara::com::SamplePtr` shall be used.

It shall return the actual number of bytes written if successful, or an `ara::core::ErrorCode` indicating the error if not successful.

The following errors from the `ara::core::raw::RawErrorDomain` is possible:

- `kStreamNotConnected`: Trying to shutdown a `RawDataStream` without an established connection.
- `kCommunicationTimeout`: No data was written until the timeout expired.

```
ara::core::Result<size_t> WriteData(ara::com::SamplePtr<uint8_t> data,  
                                      size_t length);  
ara::core::Result<size_t> WriteData(ara::com::SamplePtr<uint8_t> data,  
                                      size_t length, std::chrono::milliseconds timeout);  
] (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`

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	<i>AdaptivePlatformServiceInstance</i> (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, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i> , UploadablePackageElement			
Subclasses	<i>ProvidedApServiceInstance</i> , <i>RequiredApServiceInstance</i>			
Attribute	Type	Mult.	Kind	Note
e2eEvent ProtectionProps	<i>End2EndEvent ProtectionProps</i>	*	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	<i>End2EndMethod ProtectionProps</i>	*	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	<i>ServiceInterface ElementSecureCom Config</i>	*	aggr	Configuration settings to secure the communication of ServiceInterface elements. Tags: atp.Status=draft
serviceInterface Deployment	<i>ServiceInterface Deployment</i>	0..1	ref	Reference to a ServiceInterfaceDeployment that identifies the ServiceInterface that is represented by the Service Instance. Tags: atp.Status=draft

Table A.1: AdaptivePlatformServiceInstance

Class	Allocator			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the ability to take influence on the way objects are allocated in memory, for example it can be controlled whether an objects is allocated on the heap or on the stack. Tags: atp.Status=draft atp.recommendedPackage=Allocators			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
namespace (ordered)	<i>SymbolProps</i>	*	aggr	This aggregation allows for the definition of a namespace of an Allocator. Tags: atp.Status=draft

Table A.2: Allocator

Class	ApApplicationError			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to formally specify the semantics of an application error on the AUTOSAR adaptive platform Tags: atp.Status=draft atp.recommendedPackage=ApplicationErrors			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i> , UploadablePackageElement			
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.
errorDomain	ApApplicationError Domain	1	ref	This reference represents the error domain of the Ap ApplicationError. Tags: atp.Status=draft

Table A.3: 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, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i> , UploadablePackageElement			
Attribute	Type	Mult.	Kind	Note
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.

Table A.4: ApApplicationErrorDomain

Class	ApSomeipTransformationProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::SerializationProperties			
Note	SOME/IP serialization properties. Tags: atp.Status=draft			
Base	ARObject, <i>Identifiable</i> , MultilanguageReferrable, <i>Referrable</i> , TransformationProps			
Attribute	Type	Mult.	Kind	Note
alignment	PositiveInteger	0..1	attr	Specifies the alignment of dynamic data in the serialized data stream. The alignment is specified in Bits.
byteOrder	ByteOrderEnum	0..1	attr	Specifies the byte order of data in the serialized data stream.





Class	ApSomeipTransformationProps			
implements LegacyString Serialization	Boolean	0..1	attr	<p>This attribute indicates that Strings in the SOME/IP message shall NOT be serialized according to the SOME/IP specification for Strings.</p> <p>If this attribute is set to true, BOM and null-termination shall NOT be added in the serialization for Strings in the payload.</p> <p>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.</p> <p>NOTE! This attribute is not future safe, and will be removed in an upcoming AUTOSAR release!</p>
isDynamic LengthFieldSize	Boolean	0..1	attr	<p>This attribute represents the ability to control the setting of the wire type for TLV encoding.</p> <p>If the attribute is set to True then wire type 5-7 shall be used.</p> <p>If the attribute does not exist or is set to False then wire type 4 shall be used.</p>
session Handling	SOMEIPTransformer SessionHandlingEnum	0..1	attr	Defines whether the SOME/IP transformer shall use session handling for Sender/Receiver communication.
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.
sizeOfString LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a String. It describes the size of the length field (in Bytes) that will be put in front of the String in the SOME/IP message.
sizeOfStruct LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of an Struct. It describes the size of the length field (in Bytes) that will be put in front of the Struct in the SOME/IP message.
sizeOfUnion LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a Union. It describes the size of the length field (in Bytes) that will be put in front of the Union in the SOME/IP message.
sizeOfUnion TypeSelector Field	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a Union. It describes the size of the type selector field (in Bytes) that will be put in front of the Union in the SOME/IP message.
stringEncoding	BaseTypeEncoding String	0..1	attr	Configures the encoding for SOME/IP serialization for the referenced dataPrototype in case of an String.

Table A.5: 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





Class	ApplicationArrayDataType			
Base	<i>ARElement, ARObject, ApplicationCompositeDataType, ApplicationDataType, AtpBlueprint, Atp Blueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, Multilanguage Referrable, PackageableElement, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArray Element	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.6: ApplicationArrayDataType

Class	ApplicationDataType (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	<p>ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake.</p> <p>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.</p> <p>It should be possible to model the application level aspects of a VFB system by using ApplicationData Types only.</p>			
Base	<i>ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable</i>			
Subclasses	<i>ApplicationCompositeDataType, ApplicationPrimitiveDataType</i>			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.7: 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	<i>ARObject, Identifiable, MultilanguageReferrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
errorCode	Integer	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.8: ApplicationError

Class	ApplicationPrimitiveDataType
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes
Note	A primitive data type defines a set of allowed values. Tags: atp.recommendedPackage=ApplicationDataTypes





Class	ApplicationPrimitiveDataType			
Base	ARElement, ARObject, ApplicationDataType , AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType , CollectableElement, Identifiable , MultilanguageReferrable, PackageableElement, Referrable			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.9: 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, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType , CollectableElement, Identifiable , MultilanguageReferrable, PackageableElement, Referrable			
Attribute	Type	Mult.	Kind	Note
element (ordered)	ApplicationRecordElement	1..*	aggr	<p>Specifies an element of a record.</p> <p>The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordData Type.</p> <p>Stereotypes: atpVariation Tags:vh.latestBindingTime=preCompileTime</p>

Table A.10: ApplicationRecordDataType

Class	ApplicationRecordElement			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of one particular element of an application record data type.			
Base	ARObject, ApplicationCompositeElementDataPrototype, AtpFeature, AtpPrototype, DataPrototype , Identifiable , MultilanguageReferrable, Referrable			
Attribute	Type	Mult.	Kind	Note
isOptional	Boolean	0..1	attr	<p>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.</p> <p>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.</p>

Table A.11: ApplicationRecordElement

Class	ArgumentDataPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	<i>ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
direction	ArgumentDirectionEnum	1	attr	This attribute specifies the direction of the argument prototype.
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	<p>This defines how the argument type of the servers RunnableEntity is implemented.</p> <p>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.</p>

Table A.12: ArgumentDataPrototype

Enumeration	ArgumentDirectionEnum
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	<p>Use cases:</p> <ul style="list-style-type: none"> 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.13: ArgumentDirectionEnum

Class	AutosarDataPrototype (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	<i>ARObject, AtpFeature, AtpPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable</i>			
Subclasses	<i>ArgumentDataPrototype, Field, ParameterDataPrototype, PersistencyDataElement, VariableDataPrototype</i>			
Attribute	Type	Mult.	Kind	Note
type	AutosarDataType	1	tref	<p>This represents the corresponding data type.</p> <p>Stereotypes: isOfType</p>

Table A.14: AutosarDataPrototype

Class	AutosarDataType (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for user defined AUTOSAR data types for ECU software.			
Base	<i>ARElement, ARObject, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable</i>			
Subclasses	<i>AbstractImplementationDataType, ApplicationDataType</i>			
Attribute	Type	Mult.	Kind	Note
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this AutosarDataType.

Table A.15: AutosarDataType

Class	BaseType (abstract)			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This abstract meta-class represents the ability to specify a platform dependant base type.			
Base	<i>ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable</i>			
Subclasses	SwBaseType			
Attribute	Type	Mult.	Kind	Note
baseTypeDefinition	BaseTypeDefinition	1	aggr	<p>This is the actual definition of the base type.</p> <p>Tags:</p> <ul style="list-style-type: none"> xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table A.16: BaseType

Class	BaseTypeDirectDefinition			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	<i>ARObject, BaseTypeDefinition</i>			
Attribute	Type	Mult.	Kind	Note
baseTypeEncoding	BaseTypeEncodingString	1	attr	<p>This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence.</p> <p>Tags:xml.sequenceOffset=90</p>
baseTypeSize	PositiveInteger	0..1	attr	<p>Describes the length of the data type specified in the container in bits.</p> <p>Tags:xml.sequenceOffset=70</p>
byteOrder	ByteOrderEnum	0..1	attr	<p>This attribute specifies the byte order of the base type.</p> <p>Tags:xml.sequenceOffset=110</p>





Class	BaseTypeDirectDefinition			
memAlignment	PositiveInteger	0..1	attr	<p>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".</p> <p>Tags:xml.sequenceOffset=100</p>
native Declaration	NativeDeclarationString	0..1	attr	<p>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</p> <p>BaseType with shortName: "MyUnsignedInt" native Declaration: "unsigned short"</p> <p>Results in <code>typedef unsigned short MyUnsignedInt;</code></p> <p>If the attribute is not defined the referring Implementation DataTypes will not be generated as a typedef by RTE.</p> <p>If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseType Size.</p> <p>This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems.</p> <p>Tags:xml.sequenceOffset=120</p>

Table A.17: 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.18: ByteOrderEnum

Class	ClientServerOperation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	<i>ARObject, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, MultilanguageReferrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation Stereotypes: atpVariation Tags: vh.latestBindingTime=blueprintDerivationTime
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.19: 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	<i>ARElement, ARObject, CollectableElement, Grant, Identifiable, MultilanguageReferrable, Packageable Element, Referrable</i>			
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
serviceInstance	AdaptivePlatform ServiceInstance	1	ref	This reference identifies the applicable AdaptivePlatform ServiceInstance for which the grant applies. Tags: atp.Status=draft

Table A.20: ComEventGrant

Class	ComFieldGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	<p>This meta-class represents the ability to grant access to a ServiceInterface.field.</p> <p>Tags: atp.Status=draft atp.recommendedPackage=Grants</p>			
Base	<i>ARElement, AROObject, CollectableElement, Grant, Identifiable, MultilanguageReferrable, Packageable Element, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
design	ComFieldGrantDesign	0..1	ref	<p>This reference identifies the ComFieldGrantDesign that the enclosing ComFieldGrant was created from.</p> <p>Stereotypes: atpUriDef</p> <p>Tags:atp.Status=draft</p>
role	FieldAccessEnum	1	attr	<p>This attribute provides the ability to further specify the access to the ServiceInterface.field.</p>
service Deployment	ServiceField Deployment	1	ref	<p>This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies.</p> <p>Tags:atp.Status=draft</p>
serviceInstance	AdaptivePlatform ServiceInstance	1	ref	<p>This reference identifies the applicable AdaptivePlatform ServiceInstance for which the grant applies.</p> <p>Tags:atp.Status=draft</p>

Table A.21: ComFieldGrant

Class	ComFindServiceGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	<p>This meta-class represents the ability to grant the finding a service.</p> <p>Tags: atp.Status=draft atp.recommendedPackage=Grants</p>			
Base	<i>ARElement, AROObject, CollectableElement, Grant, Identifiable, MultilanguageReferrable, Packageable Element, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
design	ComFindServiceGrant Design	0..1	ref	<p>This reference identifies the ComFindServiceGrantDesign that the enclosing ComFindServiceGrant was created from.</p> <p>Stereotypes: atpUriDef</p> <p>Tags:atp.Status=draft</p>
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	<p>This reference identifies the AdaptivePlatformService Instances for which the grant applies.</p> <p>Tags:atp.Status=draft</p>

Table A.22: ComFindServiceGrant

Class	ComMethodGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	<p>This meta-class represents the ability to grant access to a ServiceInterface.method.</p> <p>Tags: atp.Status=draft atp.recommendedPackage=Grants</p>			
Base	ARElement, AROObject, CollectableElement, Grant, Identifiable, MultilanguageReferrable, Packageable Element, Referrable			
Attribute	Type	Mult.	Kind	Note
design	ComMethodGrant Design	0..1	ref	<p>This reference identifies the ComMethodGrantDesign that the enclosing ComMethodGrant was created from.</p> <p>Stereotypes: atpUriDef</p> <p>Tags:atp.Status=draft</p>
service Deployment	ServiceMethod Deployment	1	ref	<p>This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies.</p> <p>Tags:atp.Status=draft</p>
serviceInstance	AdaptivePlatform ServiceInstance	1	ref	<p>This reference identifies the applicable AdaptivePlatform ServiceInstance for which the grant applies.</p> <p>Tags:atp.Status=draft</p>

Table A.23: ComMethodGrant

Class	ComOfferServiceGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	<p>This meta-class represents the ability to grant the offering of a service.</p> <p>Tags: atp.Status=draft atp.recommendedPackage=Grants</p>			
Base	ARElement, AROObject, CollectableElement, Grant, Identifiable, MultilanguageReferrable, Packageable Element, Referrable			
Attribute	Type	Mult.	Kind	Note
design	ComOfferServiceGrant Design	0..1	ref	<p>This reference identifies the ComOfferServiceGrant Design that the enclosing ComOfferServiceGrant was created from.</p> <p>Stereotypes: atpUriDef</p> <p>Tags:atp.Status=draft</p>
serviceInstance	AdaptivePlatform ServiceInstance	1	ref	<p>This reference identifies the AdaptivePlatformService Instances for which the grant applies.</p> <p>Tags:atp.Status=draft</p>

Table A.24: ComOfferServiceGrant

Class	CompuConst			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the fact that the value of a computation method scale is constant.			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note





Class	CompuConst			
compuConst ContentType	CompuConstContent	1	aggr	<p>This is the actual content of the constant compu method scale.</p> <p>Tags:</p> <ul style="list-style-type: none"> xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=10 xml.typeElement=false xml.typeWrapperElement=false

Table A.25: CompuConst

Class	CompuConstTextContent			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the textual content of a scale.			
Base	<i>ARObject, CompuConstContent</i>			
Attribute	Type	Mult.	Kind	Note
vt	VerbatimString	1	attr	This represents a textual constant in the computation method.

Table A.26: CompuConstTextContent

Class	CompuMethod			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	<p>This meta-class represents the ability to express the relationship between a physical value and the mathematical representation.</p> <p>Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant.</p> <p>Tags:atp.recommendedPackage=CompuMethods</p>			
Base	<i>ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
compulInternal ToPhys	Compu	0..1	aggr	<p>This specifies the computation from internal values to physical values.</p> <p>Tags:xml.sequenceOffset=80</p>
compuPhysTo Internal	Compu	0..1	aggr	<p>This represents the computation from physical values to the internal values.</p> <p>Tags:xml.sequenceOffset=90</p>
displayFormat	DisplayFormatString	0..1	attr	<p>This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools.</p> <p>Tags:xml.sequenceOffset=20</p>
unit	Unit	0..1	ref	<p>This is the physical unit of the Physical values for which the CompuMethod applies.</p> <p>Tags:xml.sequenceOffset=30</p>

Table A.27: CompuMethod

Class	CompuRationalCoeffs			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to express a rational function by specifying the coefficients of nominator and denominator.			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
compu Denominator	CompuNominator Denominator	1	aggr	This is the denominator of the expression. Tags: xml.sequenceOffset=30
compu Numerator	CompuNominator Denominator	1	aggr	This is the numerator of the rational expression. Tags: xml.sequenceOffset=20

Table A.28: CompuRationalCoeffs

Class	CompuScale			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to specify one segment of a segmented computation method.			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
compulnverse Value	CompuConst	0..1	aggr	This is the inverse value of the constraint. This supports the case that the scale is not reversible per se. Tags: xml.sequenceOffset=60
compuScale Contents	CompuScaleContents	0..1	aggr	This represents the computation details of the scale. Tags: xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=70 xml.typeElement=false xml.typeWrapperElement=false
desc	MultiLanguageOverview Paragraph	0..1	aggr	<desc> represents a general but brief description of the object in question. Tags: xml.sequenceOffset=30
lowerLimit	Limit	0..1	attr	This specifies the lower limit of the scale. Stereotypes: atpVariation Tags: vh.latestBindingTime=preCompileTime xml.sequenceOffset=40
mask	PositiveInteger	0..1	attr	In difference to all the other computational methods every COMPU-SCALE will be applied including the bit MASK. Therefore it is allowed for this type of COMPU-METHOD, that COMPU-SCALES overlap. To calculate the string reverse to a value, the string has to be split and the according value for each substring has to be summed up. The sum is finally transmitted. The processing has to be done in order of the COMPU-SCALE elements. Tags: xml.sequenceOffset=35
shortLabel	Identifier	0..1	attr	This element specifies a short name for the particular scale. The name can for example be used to derive a programming language identifier. Tags: xml.sequenceOffset=20





Class	CompuScale			
Attribute	Type	Mult.	Kind	Note
symbol	CIdentifier	0..1	attr	<p>The symbol, if provided, is used by code generators to get a C identifier for the CompuScale. The name will be used as is for the code generation, therefore it needs to be unique within the generation context.</p> <p>Tags: xml.sequenceOffset=25</p>
upperLimit	Limit	0..1	attr	<p>This specifies the upper limit of a of the scale.</p> <p>Stereotypes: atpVariation</p> <p>Tags: vh.latestBindingTime=preCompileTime xml.sequenceOffset=50</p>

Table A.29: CompuScale

Class	CompuScales			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to stepwise express a computation method.			
Base	ARObject, CompuContent			
Attribute	Type	Mult.	Kind	Note
compuScale (ordered)	CompuScale	*	aggr	<p>This represents one scale within the compu method. Note that it contains a Variationpoint in order to support blueprints of enumerations.</p> <p>Stereotypes: atpVariation</p> <p>Tags: vh.latestBindingTime=blueprintDerivationTime xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=40 xml.typeElement=false xml.typeWrapperElement=false</p>

Table A.30: CompuScales

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
arraySize	PositiveInteger	0..1	attr	<p>This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics.</p> <p>Stereotypes: atpVariation</p> <p>Tags: vh.latestBindingTime=preCompileTime</p>





Class	CpplImplementationDataType (abstract)			
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CpplImplementationDataType. Tags: atp.Status=draft
subElement (ordered)	CpplImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CpplImplementationDataType 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 CpplImplementationDataType is contributed to the language binding.
typeReference	CpplImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef). Tags: atp.Status=draft

Table A.31: CpplImplementationDataType

Class	CpplImplementationDataTypeElement			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CpplImplementationDataType			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. A CpplImplementationDataTypeElement is used to represent an element of a structure, defining its type. Tags: atp.Status=draft			
Base	<i>ARObject, AbstractImplementationDataTypeElement, AtpClassifier, AtpFeature, AtpStructureElement, CpplImplementationDataTypeContextTarget, Identifiable, MultilanguageReferrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
isOptional	Boolean	0..1	attr	<p>This attribute represents the ability to declare the enclosing CpplImplementationDataTypeElement as optional. This means that, at runtime, the Cpp ImplementationDataTypeElement may or may not have a valid value and shall therefore be ignored.</p> <p>The underlying runtime software provides means to set the CpplImplementationDataTypeElement as not valid at the sending end of a communication and determine its validity at the receiving end.</p>
typeReference	CpplImplementation DataTypeElement Qualifier	0..1	aggr	<p>This aggregation defines the type of the Cpp ImplementationDataTypeElement and determines whether in C++ the CpplImplementationDataTypeElement is defined inside or outside of the enclosing Cpp ImplementationDataType.</p> <p>Tags:atp.Status=draft</p>

Table A.32: CpplImplementationDataTypeElement

Class	CppImplementationDataTypeElementQualifier			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This element qualifies the typeReference of the CppImplementationDataTypeElement to the CppImplementationDataType.			
	Tags: atp.Status=draft			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
inplace	Boolean	0..1	attr	This attribute defines whether the member type of the CppImplementationDataTypeElement in C++ is an embedded type element inside of the enclosing struct (true) or whether the type declaration is defined outside of the struct.
typeReference	CppImplementation DataType	1	ref	This reference defines a type reference. Tags: atp.Status=draft

Table A.33: CppImplementationDataTypeElementQualifier

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 CppTemplateArgument.
inplace	Boolean	0..1	attr	This attribute specifies whether the shortName of the referenced templateType is used in the code generation and the type declaration is defined outside of the enclosing CppImplementationDataType (true) or whether the type definition is embedded inside of the enclosing CppImplementationDataType and the shortName is ignored (false).
templateType	CppImplementation DataType	0..1	ref	This reference identifies the data type of the specific template argument required for the language binding. Tags: atp.Status=draft

Table A.34: CppTemplateArgument

Class	CustomCppImplementationDataType			
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 custom implementation that is declared in the configured header file. The Short Name of this CustomCppImplementationDataType defines the Class-Name of the custom implementation.			
	Tags: atp.Status=draft atp.recommendedPackage=CppImplementationDataTypes			





Class	CustomCppImplementationDataType			
Base	ARElement, ARObject, AbstractImplementationDataType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, CppImplementationDataType, CppImplementationDataTypeContextTarget, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	Type	Mult.	Kind	Note
headerFile	String	1	attr	Configuration of the Header File with the custom class declaration.

Table A.35: CustomCppImplementationDataType

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
swDataDefProps	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table A.36: DataPrototype

Class	DataTypeMap			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing AbstractImplementationDataType.			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
applicationDataType	ApplicationDataType	1	ref	This is the corresponding ApplicationDataType
implementationDataType	AbstractImplementationDataType	1	ref	This is the corresponding AbstractImplementationDataType.

Table A.37: DataTypeMap

Class	DataTypeMappingSet			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents a list of mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups. Tags: atp.recommendedPackage=DataTypeMappingSets			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	Type	Mult.	Kind	Note
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an ApplicationDataType and its AbstractImplementationDataType.





Class	DataTypeMappingSet			
modeRequestTypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an ModeDeclarationGroup and its AbstractImplementationData Type.

Table A.38: DataTypeMappingSet

Class	DdsEventDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for an Event. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable , ServiceEventDeployment			
Attribute	Type	Mult.	Kind	Note
topicName	DDSIDentifier	1	attr	Name of the DDS Topic associated with the Event. Tags: atp.Status=draft
transportProtocol	TransportLayerProtocol Enum	1..*	attr	This attribute defines over which Transport Layer Protocol(s) this event is intended to be sent. Tags: atp.Status=draft

Table A.39: DdsEventDeployment

Class	DdsEventQosProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Configuration properties of the Event using DDS as the underlying network binding. Tags: atp.ManifestKind=ServiceInstanceManifest 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.40: 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			
Attribute	Type	Mult.	Kind	Note
get	DdsMethodDeployment	0..1	aggr	This aggregation represents the setting of the get method. Tags: atp.Status=draft





Class	DdsFieldDeployment			
notifier	DdsEventDeployment	0..1	aggr	This aggregation represents the settings of the notifier. Tags: atp.Status=draft
set	DdsMethodDeployment	0..1	aggr	This aggregation represents the settings of the set method. Tags: atp.Status=draft

Table A.41: 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.42: DdsFieldQosProps

Class	DdsMethodDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for a Method. Tags: atp.Status=draft			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable , ServiceMethodDeployment			
Attribute	Type	Mult.	Kind	Note
ddsRpcService	DdsRpcService Deployment	0..1	ref	Configuration of the DDS-RPC service providing access to the method when using DDS as the underlying network binding. Tags: atp.Status=draft
transport Protocol	TransportLayerProtocol Enum	1..*	attr	This attribute defines over which Transport Layer Protocol(s) this method is intended to be sent. Tags: atp.Status=draft

Table A.43: DdsMethodDeployment

Class	DdsMethodQosProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Configuration properties of the Method that handles method request/replies when using DDS as the underlying network binding. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, DdsQosProps			





Class	DdsMethodQosProps			
Attribute	Type	Mult.	Kind	Note
method	ServiceMethod Deployment	1	ref	Reference to the method. Tags: atp.Status=draft

Table A.44: DdsMethodQosProps

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.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInstances			
Base	ARElement, AROObject, AdaptivePlatformServiceInstance, CollectableElement, DdsQosProps, DdsServiceInstanceProps, Identifiable, MultilanguageReferrable, PackageableElement, ProvidedApServiceInstance, Referrable, UploadablePackageElement			
Attribute	Type	Mult.	Kind	Note
eventQosProps	DdsEventQosProps	*	aggr	List of configuration properties for the Events that are provided by the Service Instance. Tags: atp.Status=draft
fieldGetSetQosProps	DdsFieldQosProps	*	aggr	List of configuration properties for the DDS-RPC service that provides access to the field getters/setters of 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
methodQosProps	DdsMethodQosProps	*	aggr	List of configuration properties for the DDS-RPC service that provides the methods of the Service Instance. Tags: atp.Status=draft
serviceInstanceId	PositiveInteger	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.45: 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, DdsMethodQosProps, DdsServiceInstanceProps			
Attribute	Type	Mult.	Kind	Note





Class	DdsQosProps (abstract)			
qosProfile	String	0..1	attr	<p>Identifies a group of QoS Policies that apply to the DDS entities associated with the event, method, field, or the service instance.</p> <p>Tags:atp.Status=draft</p>

Table A.46: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInstances			
Base	<i>ARElement, ARObject, AdaptivePlatformServiceInstance, CollectableElement, DdsQosProps, DdsServiceInstanceProps, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, RequiredApServiceInstance, UploadablePackageElement</i>			
Attribute	Type	Mult.	Kind	Note
blacklistedVersion	DdsServiceVersion	*	aggr	<p>Collection of blacklisted versions.</p> <p>Tags:atp.Status=draft</p>
eventQosProps	DdsEventQosProps	*	aggr	<p>List of configuration properties for the Events that are required by the Service Instance.</p> <p>Tags:atp.Status=draft</p>
fieldGetSetQosProps	DdsFieldQosProps	*	aggr	<p>List of configuration properties for the DDS-RPC service that requires access to the field getters/setters of the service instance.</p> <p>Tags:atp.Status=draft</p>
fieldNotifierQosProps	DdsFieldQosProps	*	aggr	<p>List of configuration properties for Field notifiers that are required by the Service Instance.</p> <p>Tags:atp.Status=draft</p>
methodQosProps	DdsMethodQosProps	*	aggr	<p>List of configuration properties for the DDS-RPC service that requires access to the methods of the service instance.</p> <p>Tags:atp.Status=draft</p>
requiredServiceInstanceId	AnyServiceInstanceld	1	attr	<p>This attribute represents the ability to describe the required service instance ID.</p> <p>Tags:atp.Status=draft</p>

Table A.47: DdsRequiredServiceInstance

Class	DdsRpcServiceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	Configuration settings for a DDS-RPC service capable of providing access to the methods and field getters/setters of a service interface.			
	Tags: atp.Status=draft			





Class	DdsRpcServiceDeployment			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable			
Attribute	Type	Mult.	Kind	Note
replyTopicName	DDIdentifier	0..1	attr	Name of the DDS Reply Topic associated with the Method. Tags: atp.Status=draft
requestTopic Name	DDIdentifier	0..1	attr	Name of the DDS Request Topic associated with the Method. Tags: atp.Status=draft

Table A.48: DdsRpcServiceDeployment

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			
Attribute	Type	Mult.	Kind	Note
domainId	Integer	1	attr	This attribute identifies the DDS Domain the Service Instance shall join. Tags: atp.Status=draft
transportPlugin	String	1..*	attr	Enable a transport plug-in (e.g., sharedMemory) in the underlying DDS binding implementation. Tags: atp.Status=draft

Table A.49: DdsServiceInstanceProps

Class	DdsServiceInterfaceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for a ServiceInterface. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInterfaceDeployments			
Base	ARElement, ARObject, CollectableElement, Identifiable , MultilanguageReferrable , Packageable Element, Referrable , ServiceInterfaceDeployment , UploadablePackageElement			
Attribute	Type	Mult.	Kind	Note
ddsRpcService	DdsRpcService Deployment	*	aggr	This aggregation represents the settings of DDS-RPC services associated with a Service Interface to handle methods and field getters and setters when using DDS as the underlying network binding. Tags: atp.Status=draft





Class	DdsServiceInterfaceDeployment			
serviceInterfaceId	String	1	attr	<p>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.</p> <p>Tags:atp.Status=draft</p>

Table A.50: DdsServiceInterfaceDeployment

Class	E2EProfileConfiguration			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E			
Note	<p>This element holds E2E profile specific configuration settings.</p> <p>Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft</p>			
Base	<i>ARObject</i> , <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
clearFromValidToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.
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.
maxDeltaCounter	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.
maxErrorStateInit	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.
maxErrorStateInvalid	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.
maxErrorStateValid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_VALID.
minOkStateInit	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT.
minOkStateInvalid	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.
minOkStateValid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID.
profileName	NameToken	1	attr	Definition of the E2E profile.
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine.
windowSizeInvalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.





Class	E2EProfileConfiguration			
windowSizeValid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table A.51: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	<i>ARObject</i> , <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
dataId (ordered)	PositiveInteger	*	attr	<p>This represents a unique numerical identifier for the referenced event or field notifier that is included in the CRC calculation.</p> <p>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.</p>
dataLength	PositiveInteger	0..1	attr	Length of payload including E2E header in bits.
dataUpdatePeriod	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.
e2eProfileConfiguration	E2EProfileConfiguration	0..1	ref	<p>Reference to E2E profile configuration settings that are valid to protect the referenced event or field notifier.</p> <p>Tags:atp.Status=draft</p>
event	ServiceEventDeployment	0..1	ref	<p>Reference to an event that is protected by the E2E profile.</p> <p>Tags:atp.Status=draft</p>
maxDataLength	PositiveInteger	0..1	attr	Maximum length of payload including E2E header in bits.
minDataLength	PositiveInteger	0..1	attr	Minimum length of payload including E2E header in bits.

Table A.52: End2EndEventProtectionProps

Class	End2EndMethodProtectionProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E			
Note	This element allows to protect a method, a field setter or a field getter with an E2E profile. Tags: atp.Status=draft			
Base	<i>ARObject</i>			
Attribute	Type	Mult.	Kind	Note





Class	End2EndMethodProtectionProps			
dataId (ordered)	PositiveInteger	*	attr	<p>This represents a numerical identifier that is included in the CRC calculation. This dataId is used for call and response.</p> <p>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.</p>
dataLength	PositiveInteger	0..1	attr	Length of payload including E2E header in bits.
dataUpdate Period	TimeValue	0..1	attr	This attribute describes the period in which the applications are assumed to process E2E-protected messages. The middleware does not use this attribute at all.
e2eProfile Configuration	E2EProfileConfiguration	0..1	ref	<p>Reference to E2E profile configuration settings that are valid to protect the referenced method, field getter or field setter.</p> <p>Tags:atp.Status=draft</p>
maxDataLength	PositiveInteger	0..1	attr	Maximum length of payload including E2E header in bits.
method	ServiceMethod Deployment	0..1	ref	<p>Reference to a method, a field getter or a field setter that is protected by the E2E profile.</p> <p>Tags:atp.Status=draft</p>
minDataLength	PositiveInteger	0..1	attr	Minimum length of payload including E2E header in bits.

Table A.53: End2EndMethodProtectionProps

Class	EndToEndTransformationComSpecProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	The class EndToEndTransformationComSpecProps specifies port specific configuration properties for EndToEnd transformer attributes.			
Base	ARObject, Describable, TransformationComSpecProps			
Attribute	Type	Mult.	Kind	Note
clearFromValid ToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.
disableEndTo EndCheck	Boolean	1	attr	Disables/Enables the E2E check. The E2Eheader is removed from the payload independent from the setting of this attribute.
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	<p>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.</p> <p>The minimum value is 0.</p>
maxErrorState Invalid	PositiveInteger	0..1	attr	<p>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.</p> <p>The minimum value is 0.</p>





Class	EndToEndTransformationComSpecProps			
maxErrorStateValid	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.
minOkStateInvalid	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.
minOkStateValid	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.
windowSizeInvalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.
windowSizeValid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table A.54: EndToEndTransformationComSpecProps

Class	EthernetCommunicationConnector			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet specific attributes to the CommunicationConnector. Tags: atp.ManifestKind=MachineManifest			
Base	ARObject, CommunicationConnector, Identifiable , MultilanguageReferrable , Referrable			
Attribute	Type	Mult.	Kind	Note
maximumTransmissionUnit	PositiveInteger	0..1	attr	This attribute specifies the maximum transmission unit in bytes.
networkEndpoint	NetworkEndpoint	*	ref	NetworkEndpoints
pathMtuEnabled	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.
pncFilterDataMask	PositiveUnlimitedInteger	0..1	attr	Bit mask for Ethernet Payload used to configure the Ethernet Transceiver for partial network wakeup.
unicastNetworkEndpoint	NetworkEndpoint	0..1	ref	Network Endpoint that defines the IPAddress of the machine. Tags: atp.Status=draft

Table A.55: EthernetCommunicationConnector

Class	EthernetRawDataStreamMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class represents the ability to map a PortPrototype to a Ethernet-based communication channel. Tags: atp.Status=draft atp.recommendedPackage=SocketRawDataStreamMappings			
Base	<i>ARElement, AROObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, RawDataStreamMapping, Referrable, UploadablePackageElement</i>			
Attribute	Type	Mult.	Kind	Note
communication Connector	EthernetCommunication Connector	1	ref	This attribute represents the CommunicationConnector taken for socket-based data communication. Tags: atp.Status=draft
multicastUdp Port	PositiveInteger	0..1	attr	This attribute describes the UDP Port used for multicast raw data stream transmission.
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.
tcpPort	PositiveInteger	0..1	attr	This attribute describes the TCP port used for the raw data streaming.
tlsSecureCom Props	TlsSecureComProps	0..1	aggr	This aggregation provides the ability to define TLS-related properties for the enclosing SocketRawDataStream Mapping. Tags: atp.Status=draft
udpPort	PositiveInteger	0..1	attr	This attribute describes the UDP Port used for the raw data streaming.

Table A.56: EthernetRawDataStreamMapping

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	<i>ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, Multilanguage Referrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
hasGetter	Boolean	1	attr	This attribute controls whether read access is foreseen to this field.
hasNotifier	Boolean	1	attr	This attribute controls whether a notification semantics is foreseen to this field.
hasSetter	Boolean	1	attr	This attribute controls whether write access is foreseen to this field.

Table A.57: 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
getterSetter	Access to getter and setter of the field Tags: atp.EnumerationLiteralIndex=2
setter	Access to the setter of the Field. Tags: atp.EnumerationLiteralIndex=1

Table A.58: 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.59: 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
ipSecConfigProps	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.60: IPSecConfig

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





Class	IPSecRule			
direction	CommunicationDirectionType	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	1	attr	Header type specifying the IPsec security mechanism.
ike Authentication Method	IkeAuthenticationMethodEnum	1	attr	This attribute defines the IKE authentication method that is used locally and is expected on the remote side.
ipProtocol	IPsecIpProtocolEnum	1	attr	This attribute defines the relevant IP protocol used in the Security Policy Database (SPD) entry.
localPortRangeEnd	PositiveInteger	0..1	attr	<p>This attribute restricts the traffic monitoring to tcp and defines a end value for the local port range.</p> <p>If this attribute is not set then this rule shall be effective for all local ports.</p> <p>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.</p>
localPortRangeStart	PositiveInteger	0..1	attr	<p>This attribute restricts the traffic monitoring to tcp and defines a start value for the local port range.</p> <p>If this attribute is not set then this rule shall be effective for all local ports.</p> <p>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.</p>
mode	IPsecModeEnum	1	attr	This attribute defines the type of the connection.
policy	IPsecPolicyEnum	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.
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".
remoteIpEndpointAddress	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.
remotePortRangeEnd	PositiveInteger	0..1	attr	<p>This attribute restricts the traffic monitoring to tcp and defines a end value for the remote port range.</p> <p>If this attribute is not set then this rule shall be effective for all local ports.</p> <p>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.</p>
remotePortRangeStart	PositiveInteger	0..1	attr	<p>This attribute restricts the traffic monitoring to tcp and defines a start value for the remote port range.</p> <p>If this attribute is not set then this rule shall be effective for all local ports.</p>





Class	IPSecRule			
	△ 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.61: IPSecRule

Class	ISignalToIPduMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	An ISignalToIPduMapping describes the mapping of ISignals to ISignalIPdus and defines the position of the ISignal within an ISignalIPdu.			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable			
Attribute	Type	Mult.	Kind	Note
iSignal	ISignal	0..1	ref	<p>Reference to a ISignal that is mapped into the ISignal IPdu.</p> <p>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.</p>
iSignalGroup	ISignalGroup	0..1	ref	<p>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.</p> <p>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.</p>
packingByteOrder	ByteOrderEnum	0..1	attr	<p>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).</p> <p>For an ISignalGroup the packingByteOrder is irrelevant and shall be ignored.</p>
startPosition	Integer	0..1	attr	<p>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.</p> <p>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.</p> <p>If a mapping for the ISignalGroup is defined, this attribute is irrelevant and shall be ignored.</p>





Class	ISignalToIPduMapping			
transferProperty	TransferPropertyEnum	0..1	attr	Defines how the referenced ISignal contributes to the send triggering of the ISignalPdu.
update IndicationBit Position	Integer	0..1	attr	<p>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 SignallPdu. For Signals of a ISignalGroup this attribute is irrelevant and shall be ignored.</p> <p>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 ISignalPdu still undergoes a change.</p> <p>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.</p>

Table A.62: ISignalToIPduMapping

Class	ISignalTriggering			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	A ISignalTriggering allows an assignment of ISignals to physical channels.			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable			
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	<p>References to the ISignalPort on every ECU of the system which sends and/or receives the ISignal.</p> <p>References for both the sender and the receiver side shall be included when the system is completely defined.</p>

Table A.63: ISignalTriggering

Class	Identifiable (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	ARObject, MultilanguageReferrable, Referrable			
Subclasses	ARPackage, AbstractEvent, AbstractImplementationDataTypeElement, AbstractServiceInstance, AbstractSignalBasedToSignalTriggeringMapping, AdaptiveModuleInstantiation, AdaptiveSwcInternalBehavior, ApplicationEndpoint, ApplicationError , ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpFeature, AutosarOperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModuleDependency, BuildActionEntity, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation , Code, CollectableElement, ComManagementMapping, CommConnectorPort, CommunicationConnector, CommunicationController, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, CouplingPortStructuralElement, CryptoKeySlot, CryptoServiceMapping, DataPrototypeGroup, DataTransformation, DdsRpcServiceDeployment , DependencyOnArtifact, DeterministicClientResourceNeeds, DiagEventDebounceAlgorithm, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticFunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, DiagnosticRoutineSubfunction, DltArgument, DltLogChannel, DltMessage, DolpInterface, DolpLogicAddress, E2EProfileConfiguration , ECUMapping, EOCExecutableEntityRefAbstract, EcuPartition, EcucContainerValue, EcucDefinitionElement, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEventProtectionProps , EndToEndProtection, EventMapping, ExclusiveArea, ExecutableEntity, ExecutionTime, FMAtributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForgetMapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, FrameTriggering, GeneralParameter, GlobalTimeGateway, GlobalTimeMaster, GlobalTimeSlave, HealthChannel, HeapUsage, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPsecRule, IPv6ExtHeaderFilterList, ISignalToTpMapping , ISignalTriggering , IdentCaption, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, MemorySection, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint , NmCluster, NmNode, NvBlockDescriptor, PackageableElement, ParameterAccess, PduToFrameMapping, PduTriggering, PerInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, PhmActionItem, PhmActionList, PhmLogicalExpression, PhmRule, PhmSupervision, PhysicalChannel, PortGroup, PortInterfaceMapping, PossibleErrorReaction, ProcessDesignToMachineDesignMapping, ProcessToMachineMapping, Processor, ProcessorCore, PskeyIdentityToKeySlotMapping, RawDataStreamMethodDeployment , ResourceConsumption, ResourceGroup, RestAbstractEndpoint, RestElementDef, RestResourceDef, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, SdgAttribute, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureComProps , SecureCommunicationAuthenticationProps, SecureCommunicationDeployment, SecureCommunicationFreshnessProps, ServerCallPoint, ServiceEventDeployment , ServiceFieldDeployment, ServiceInstanceToSignalMapping, ServiceInterfaceElementMapping, ServiceInterfaceElementSecureComConfig, ServiceInterfaceMapping, ServiceMethodDeployment , ServiceNeeds, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, SpecElementReference, StackUsage, StartupConfig, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, SystemMemoryUsage, TcpOptionFilterList, TimeBaseResource, TimingCondition, TimingConstraint, TimingDescription, TimingExtensionResource, TimingModelInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, TracedFailure, TransformationProps, TransformationPropsToServiceInterfaceElementMapping, TransformationTechnology, Trigger, UcmDescription, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	Type	Mult.	Kind	Note
adminData	AdminData	0..1	agr	This represents the administrative data for the identifiable object. Tags: xml.sequenceOffset=-40





Class	Identifiable (abstract)			
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	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". Tags: xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. Tags: xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. Tags: xml.attribute=true

Table A.64: Identifiable

Class	ImplementationDataType
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. Tags: atp.recommendedPackage=ImplementationDataTypes
Base	<i>ARElement, ARObject, AbstractImplementationDataType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable</i>





Class	ImplementationDataType			
Attribute	Type	Mult.	Kind	Note
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWithOptionalElement	Boolean	0..1	attr	<p>This attribute is only valid if the attribute category is set to STRUCTURE.</p> <p>If set to True, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.</p>
subElement(ordered)	ImplementationDataTypeElement	*	aggr	<p>Specifies an element of an array, struct, or union data type.</p> <p>The aggregation of ImplementationDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure.</p> <p>Stereotypes: atpVariation Tags:vh.latestBindingTime=preCompileTime</p>
symbolProps	SymbolProps	0..1	aggr	<p>This represents the SymbolProps for the Implementation DataType.</p> <p>Stereotypes: atpSplittable Tags:atp.Splitkey=shortName</p>
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table A.65: ImplementationDataType

Class	ImplementationDataTypeElement			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	<p>Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated.</p> <p>This element either consists of further subElements or it is further defined via its swDataDefProps.</p> <p>There are several use cases within the system of ImplementationDataTypes for such a local declaration:</p> <ul style="list-style-type: none"> • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element. 			
Base	<i>ARObject, AbstractImplementationDataTypeElement, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, MultilanguageReferable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
arraySize	PositiveInteger	0..1	attr	<p>The existence of this attribute (if bigger than 0) defines the size of an array and declares that this ImplementationDataTypeElement represents the type of each single array element.</p> <p>Stereotypes: atpVariation Tags:vh.latestBindingTime=preCompileTime</p>
arraySizeHandling	ArraySizeHandlingEnum	0..1	attr	The way how the size of the array is handled in case of a variable size array.
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls the meaning of the value of the array size.





Class	ImplementationDataTypeElement			
isOptional	Boolean	0..1	attr	<p>This attribute represents the ability to declare the enclosing ImplementationDataTypeElement as optional. This means that, at runtime, the ImplementationDataType Element may or may not have a valid value and shall therefore be ignored.</p> <p>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.</p>
subElement (ordered)	ImplementationDataTypeElement	*	aggr	<p>Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs").</p> <p>The aggregation of ImplementationDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure.</p> <p>Stereotypes: atpVariation Tags:vh.latestBindingTime=preCompileTime</p>
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElement.

Table A.66: ImplementationDataTypeElement

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.			
Base	ARObject, Referrable			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps , SymbolicNameProps			
Attribute	Type	Mult.	Kind	Note
symbol	CIdentifier	1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table A.67: 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. Tags: atp.ManifestKind=ServiceInstanceManifest			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
initialDelayMax Value	TimeValue	1	attr	Max Value in seconds to delay randomly the first offer (if aggregated in role initialOfferBehavior by SomeipSd ServerServiceInstanceConfig) or the transmission of a find message (if aggregated in role initialFindBehavior by SomeipSdClientServiceInstanceConfig).





Class	InitialSdDelayConfig			
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.68: InitialSdDelayConfig

Class	Ipv4Configuration			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Internet Protocol version 4 (IPv4) configuration.			
Base				
Attribute	Type	Mult.	Kind	Note
assignmentPriority	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.
dnsServerAddress	Ip4AddressString	*	attr	IP addresses of preconfigured DNS servers. Tags: xml.namePlural=DNS-SERVER-ADDRESSES
ipAddressKeepBehavior	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.
ipv4AddressSource	Ipv4AddressSourceEnum	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.69: Ipv4Configuration

Class	Ipv6Configuration			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Internet Protocol version 6 (IPv6) configuration.			
Base	<i>ARObject, NetworkEndpointAddress</i>			
Attribute	Type	Mult.	Kind	Note
assignmentPriority	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.
dnsServerAddress	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)
ipAddressKeepBehavior	IpAddressKeepEnum	0..1	attr	Defines the lifetime of a dynamically fetched IP address.
ipAddressPrefixLength	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.
ipv6AddressSource	Ipv6AddressSourceEnum	0..1	attr	Defines how the node obtains its IP address.

Table A.70: Ipv6Configuration

Primitive	Limit			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes			
Note	This class represents the ability to express a numerical limit. Note that this is in fact a NumericalVariation Point but has the additional attribute intervalType. Tags: xml.xsd.customType=LIMIT-VALUE xml.xsd.pattern=(0[xX][0-9a-fA-F]+) (0[0-7]+) (0[bB][0-1]+) (([+-]?[1-9][0-9]+ (.[0-9]+)?[+-]?[0-9](.[0-9]+)?)([eE](+[+-]?[0-9]+)? \.\.0 INF -INF NaN xml.xsd.type=string			
Attribute	Type	Mult.	Kind	Note
intervalType	IntervalTypeEnum	0..1	attr	This specifies the type of the interval. If the attribute is missing the interval shall be considered as "CLOSED". Tags: xml.attribute=true

Table A.71: Limit

Class	Machine			
Package	M2::AUTOSARTemplates::AdaptivePlatform::MachineManifest			
Note	Machine that represents an Adaptive Autosar Software Stack. Tags: atp.ManifestKind=MachineManifest atp.Status=draft atp.recommendedPackage=Machines			
Base	<i>ARElement, ARObject, AtpClassifier, AtpFeature, AtpStructureElement, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
default Application Timeout	EnterExitTimeout	0..1	aggr	This aggregation defines a default timeout in the context of a given Machine with respect to the launching and termination of applications. Tags: atp.Status=draft
environment Variable	TagWithValueOptionalValue	*	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 atp.Status=draft
functionGroup	ModeDeclarationGroup Prototype	*	aggr	This aggregation represents the collection of function groups of the enclosing Machine. Stereotypes: atpSplitable; atpVariation Tags: atp.Splitkey=shortName, variationPoint.shortLabel atp.Status=draft vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	This reference is used to describe the hardware resources of the machine. Stereotypes: atpUriDef Tags: 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=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=shortName atp.Status=draft





Class	Machine			
trustedPlatform Executable LaunchBehavior	TrustedPlatform ExecutableLaunch BehaviorEnum	1	attr	This attribute controls the behavior of how authentication affects the ability to launch for each Executable.

Table A.72: 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). Tags: atp.ManifestKind=MachineManifest			
Base	ARObject, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
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.73: NetworkEndpoint

Class	<<atpPrototype>> PduToFrameMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	A PduToFrameMapping defines the composition of Pdus in each frame.			
Base	ARObject, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
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.





<<atpPrototype>> PduToFrameMapping				
Class	Type	Mult.	Kind	Note
updateIndicationBitPosition	Integer	0..1	attr	<p>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.</p> <p>Note that the exact bit position of the updateIndicationBitPosition is linked to the value of the attribute packingByteOrder because the method of finding the bit position is different for the values mostSignificantByteFirst and mostSignificantByteLast. This means that if the value of packingByteOrder is changed while the value of updateIndicationBitPosition remains unchanged the exact bit position of updateIndicationBitPosition within the enclosing Frame still undergoes a change.</p> <p>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.</p>

Table A.74: PduToFrameMapping

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	ClientServerInterface, CompositeInterface, DataInterface, DiagnosticPortInterface, ModeSwitchInterface, PersistenceInterface, PlatformHealthManagementInterface, RawDataStreamInterface, RestServiceInterface, ServiceInterface, TimeSynchronizationInterface, TriggerInterface			
Attribute	Type	Mult.	Kind	Note
namespace (ordered)	SymbolProps	*	aggr	<p>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.</p> <p>Stereotypes: atpSplittable</p> <p>Tags:</p> <ul style="list-style-type: none"> atp.Splitkey=shortName atp.Status=draft

Table A.75: PortInterface

Class	PortInterfaceToDataTypeMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			





Class	PortInterfaceToDataTypeMapping			
Note	This meta-class represents the ability to associate a PortInterface with a DataTypeMappingSet. This association is needed for the generation of header files in the scope of a single PortInterface. The association is intentionally made outside the scope of the PortInterface itself because the designers of a PortInterface most likely will not want to add details about the level of ImplementationDataType. Tags: atp.Status=draft atp.recommendedPackage=ServiceInterfaceToDataTypeMappings			
Base	<i>ARElement, AROObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement</i>			
Attribute	Type	Mult.	Kind	Note
dataTypeMappingSet	DataTypeMappingSet	1..*	ref	This represents the reference to the applicable data TypemappingSet Tags: atp.Status=draft atp.StatusComment=Reserved for adaptive platform
portInterface	PortInterface	1	ref	This represents the reference to the applicable Port Interface Tags: atp.Status=draft atp.StatusComment=Reserved for adaptive platform

Table A.76: PortInterfaceToDataTypeMapping

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	<i>ARElement, AROObject, AdaptivePlatformServiceInstance, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement</i>			
Subclasses	DdsProvidedServiceInstance , ProvidedSomeipServiceInstance , ProvidedUserDefinedServiceInstance			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.77: ProvidedApServiceInstance

Class	ProvidedServiceInstance			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	Service instances that are provided by the ECU that is connected via the ApplicationEndpoint to a CommunicationConnector.			
Base	<i>ARObject, AbstractServiceInstance, Identifiable, MultilanguageReferrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
autoAvailable	Boolean	0..1	attr	Defines that this ProvidedServiceInstance shall be offered by the service discovery at ECU start.





Class	ProvidedServiceInstance			
eventHandler	EventHandler	*	aggr	Collection of event groups provided by the Provided ServiceInstance Stereotypes: atpVariation Tags: vh.latestBindingTime=postBuild
instance Identifier	PositiveInteger	0..1	attr	Instance identifier. Can be used for e.g. service discovery to identify the instance of the service.
loadBalancing Priority	PositiveInteger	0..1	attr	Defines the value to be used for load balancing priority in the service offer. Lower value means higher priority.
loadBalancing Weight	PositiveInteger	0..1	attr	Defines the value to be used for load balancing weight in the service offer. Higher value means higher probability to be chosen.
localUnicast Address	ApplicationEndpoint	0..2	ref	The local address over which the PSI is provided (udp, tcp or both). Stereotypes: atpVariation Tags: vh.latestBindingTime=postBuild
minorVersion	PositiveInteger	0..1	attr	Minor Version of the Service that is provided by this ProvidedServiceInstance.
priority	PositiveInteger	0..1	attr	Defines the frame priority where values from 0 (best effort) to 7 (highest) are allowed.
remoteUnicast Address	ApplicationEndpoint	*	ref	This reference defines the remote addresses of service consumers. This reference shall ONLY be used if the remote address of the clients is determined from the configuration and not at runtime. Stereotypes: atpVariation Tags: vh.latestBindingTime=postBuild
sdServerConfig	SdServerConfig	0..1	aggr	Service Discovery Server configuration. Tags: atp.Status=obsolete
sdServerTimer Config	SomeipSdServer ServiceInstanceConfig	0..1	ref	Server specific configuration settings relevant for the SOME/IP service discovery. Stereotypes: atpVariation Tags: vh.latestBindingTime=postBuild
serviceIdentifier	PositiveInteger	0..1	attr	This attribute represents the ability to describe the SOME/IP service ID that is offered.

Table A.78: ProvidedServiceInstance

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.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInstances			
Base	ARElement, ARObject, AdaptivePlatformServiceInstance, CollectableElement, Identifiable, MultilanguageReferable, PackageableElement, ProvidedApServiceInstance, Referrable, Uploadable PackageElement			
Attribute	Type	Mult.	Kind	Note





Class	ProvidedSomeipServiceInstance			
capability Record (ordered)	TagWithValueOptionalValue	*	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
loadBalancing Priority	PositiveInteger	0..1	attr	This attribute is used to specify the priority in the load balancing option of SOME/IP that is added to the Offer Service. When a client searches for all service instances of a service, the client shall choose the service instance with highest priority if one is defined.
loadBalancing Weight	PositiveInteger	0..1	attr	This attribute is used to specify the weight in the load balancing option of SOME/IP that is added to the Offer Service. When a client searches for all service instances of a service, the client shall choose the service instance with highest priority if one is defined. If several service instances exist with the highest priority the service instance shall be chosen based on the weights of the service instances.
method ResponseProps	SomeipMethodProps	*	aggr	Configuration settings for individual methods that are provided by the ServiceInstance. 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
serviceInstanceId	PositiveInteger	1	attr	Identification number that is used by SOME/IP service discovery to identify the instance of the service.

Table A.79: 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.ManifestKind=ServiceInstanceManifest 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.80: ProvidedUserDefinedServiceInstance

Class	RawDataStreamGrant (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
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, AROObject, CollectableElement, Grant, <i>Identifiable</i> , MultilanguageReferrable, PackageableElement, <i>Referrable</i>			
Subclasses	EthernetRawDataStreamGrant			
Attribute	Type	Mult.	Kind	Note
design	RawDataStreamGrant Design	0..1	ref	<p>This reference identifies the RawDataStreamGrantDesign that the enclosing RawDataStreamEventGrant was created from.</p> <p>Stereotypes: atpUriDef Tags:atp.Status=draft</p>

Table A.81: RawDataStreamGrant

Class	RawDataStreamInterface			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	<p>This meta-class represents the necessary capabilities for raw data streaming, i.e. the streaming of data that do not undergo any serialization.</p> <p>The necessary capabilities are modeled by means of meta-class ClientServerOperation aggregated by the RawDataStreamInterface:</p> <ul style="list-style-type: none"> • 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 <p>Tags: atp.Status=draft atp.recommendedPackage=RawDataStreamInterfaces</p>			
Base	ARElement, AROObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, PackageableElement, <i>PortInterface</i> , <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
connect	ClientServerOperation	1	aggr	<p>This aggregation represents the capability of the enclosing RawStreamingDataInterface to set up the communication channel.</p> <p>Tags:atp.Status=draft</p>
read	ClientServerOperation	1	aggr	<p>This aggregation represents the capability of the enclosing RawStreamingDataInterface to access incoming data on the communication channel.</p> <p>Tags:atp.Status=draft</p>
shutdown	ClientServerOperation	1	aggr	<p>This aggregation represents the capability of the enclosing RawStreamingDataInterface to close the communication channel.</p> <p>Tags:atp.Status=draft</p>





Class	RawDataStreamInterface			
write	ClientServerOperation	1	aggr	<p>This aggregation represents the capability of the enclosing RawStreamingDataInterface to send data down the communication channel.</p> <p>Tags:atp.Status=draft</p>

Table A.82: RawDataStreamInterface

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	<p>This reference identifies the applicable RawDataStream Deployment.</p> <p>Tags:atp.Status=draft</p>
portPrototype	RPortPrototype	0..1	ioref	<p>Reference to a specific PortPrototype that represents the raw data stream to the application.</p> <p>Tags:atp.Status=draft</p>
process	Process	0..1	ref	<p>Reference to the Process in which the Executable that contains the SoftwareComponent and the referenced Port Prototype is executed.</p> <p>Tags:atp.Status=draft</p>

Table A.83: RawDataStreamMapping

Class	RawDataStreamMethodDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class represents the ability to define deployment-specific attributes for ClientServerOperations aggregated by a RawDataStreamInterface.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	Type	Mult.	Kind	Note
callTimeout	TimeValue	0..1	attr	<p>This attribute provides the ability to define a timeout value for a method call.</p>
method	ClientServerOperation	*	ref	<p>This reference identifies the method to which the deployment information shall apply.</p> <p>Tags:atp.Status=draft</p>

Table A.84: RawDataStreamMethodDeployment

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	<i>ARObject</i>			
Subclasses	<i>AtpDefinition</i> , <i>BswDistinguishedPartition</i> , <i>BswModuleCallPoint</i> , <i>BswModuleClientServerEntry</i> , <i>BswVariableAccess</i> , <i>CouplingPortTrafficClassAssignment</i> , <i>CpplImplementationDataWithTypeContextTarget</i> , <i>DiagnosticDebounceAlgorithmProps</i> , <i>DiagnosticEnvModeElement</i> , <i>EthernetPriorityRegeneration</i> , <i>EventHandler</i> , <i>ExclusiveAreaNestingOrder</i> , <i>HwDescriptionEntity</i> , <i>ImplementationProps</i> , <i>LinSlaveConfigIdent</i> , <i>ModeTransition</i> , <i>MultilanguageReferrable</i> , <i>NetworkConfiguration</i> , <i>NmNetworkHandle</i> , <i>PduActivation</i> , <i>RoutingGroup</i> , <i>PncMappingIdent</i> , <i>SingleLanguageReferrable</i> , <i>SoConIPdulIdentifier</i> , <i>SocketConnection</i> , <i>Bundle</i> , <i>SomeipRequiredEventGroup</i> , <i>TimeSyncServerConfiguration</i> , <i>TpConnectionIdent</i>			
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. 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.85: Referrable

Class	RequestResponseDelay			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	Time to wait before answering the query. Tags: atp.ManifestKind=ServiceInstanceManifest			
Base	<i>ARObject</i>			
Attribute	Type	Mult.	Kind	Note
maxValue	TimeValue	1	attr	Maximum allowable response delay to entries received by multicast in seconds.
minValue	TimeValue	1	attr	Minimum allowable response delay to entries received by multicast in seconds.

Table A.86: 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	<i>ARElement</i> , <i>ARObject</i> , <i>AdaptivePlatformServiceInstance</i> , <i>CollectableElement</i> , <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>PackageableElement</i> , <i>Referrable</i> , <i>UploadablePackageElement</i>			
Subclasses	<i>DdsRequiredServiceInstance</i> , <i>RequiredSomeipServiceInstance</i> , <i>RequiredUserDefinedServiceInstance</i>			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.87: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInstances			
Base	<i>ARElement, AROObject, AdaptivePlatformServiceInstance, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, RequiredApServiceInstance, Uploadable PackageElement</i>			
Attribute	Type	Mult.	Kind	Note
blacklisted Version	SomeipServiceVersion	*	aggr	Collection of blacklisted versions. Tags: atp.Status=draft
capability Record (ordered)	TagWithValueOptionalValue	*	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 Somelp 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.
requiredService InstanceId	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID. Tags: atp.Status=draft
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.

Table A.88: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInstances			
Base	<i>ARElement, AROObject, AdaptivePlatformServiceInstance, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, RequiredApServiceInstance, Uploadable PackageElement</i>			





Class	RequiredUserDefinedServiceInstance			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.89: RequiredUserDefinedServiceInstance

Class	SecOcSecureComProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	Configuration of AUTOSAR SecOC. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable , SecureComProps			
Attribute	Type	Mult.	Kind	Note
authAlgorithm	String	0..1	attr	This attribute defines the authentication algorithm used for MAC generation and verification.
authInfoTxLength	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.
freshnessValueLength	PositiveInteger	0..1	attr	This attribute defines the complete length in bits of the Freshness Value.
freshnessValueTxLength	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.
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	ARObject, Identifiable , MultilanguageReferrable , Referrable			
Subclasses	SecOcSecureComProps , TlsSecureComProps			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.91: SecureComProps

Enumeration	SerializationTechnologyEnum			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	This enumeration allows to choose a Serialization Technology. Tags:atp.Status=draft			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-





Enumeration	SerializationTechnologyEnum
Literal	Description
signalBased	Signal-Based serializer. Tags: atp.EnumerationLiteralIndex=1
someip	SOME/IP Serializer Tags: atp.EnumerationLiteralIndex=0

Table A.92: 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, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
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

Table A.93: 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, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
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.94: 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			





Class	ServiceInstanceToMachineMapping (abstract)			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i> , UploadablePackageElement			
Subclasses	DdsServiceInstanceToMachineMapping, SomeipServiceInstanceToMachineMapping, UserDefined ServiceInstanceToMachineMapping			
Attribute	Type	Mult.	Kind	Note
communication Connector	Communication Connector	0..1	ref	<p>Reference to the Machine to which the ServiceInstance is mapped.</p> <p>Tags:atp.Status=draft</p>
secOcCom PropsFor Multicast	SecOcSecureCom Props	*	ref	<p>Reference to communication security configuration settings that are valid for the udp multicast endpoint (Port + Multicast IP Address) defined by the ServiceInstanceToMachineMapping.</p> <p>Tags:atp.Status=draft</p>
secureCom PropsForTcp	SecureComProps	*	ref	<p>Reference to communication security configuration settings that are valid for the tcp unicast endpoint (Tcp Port + Unicast IP Address) defined by the Service InstanceToMachineMapping.</p> <p>Tags:atp.Status=draft</p>
secureCom PropsForUdp	SecureComProps	*	ref	<p>Reference to communication security configuration settings that are valid for the udp unicast endpoint (Udp Port + Unicast IP Address) defined by the Service InstanceToMachineMapping.</p> <p>Tags:atp.Status=draft</p>
serviceInstance	AdaptivePlatform ServiceInstance	*	ref	<p>Reference to a ServiceInstance that is mapped to the Machine.</p> <p>Tags:atp.Status=draft</p>

Table A.95: ServiceInstanceToMachineMapping

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				
Base	ARObject, <i>Identifiable</i> , MultilanguageReferrable, <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
eventElement Mapping	SignalBasedEvent ElementToISignal TriggeringMapping	*	aggr	<p>Mapping of an event or an element inside of the event to an ISignalTriggering.</p> <p>Tags:atp.Status=draft</p>
fieldMapping	SignalBasedFieldToISignalTriggering Mapping	*	aggr	<p>Mapping of a field to ISignalTriggerings.</p> <p>Tags:atp.Status=draft</p>
methodMapping	SignalBasedMethodToISignalTriggering Mapping	0..1	aggr	<p>Mapping of a method to ISignalTriggerings.</p> <p>Tags:atp.Status=draft</p>





Class	ServiceInstanceToSignalMapping			
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

Table A.96: ServiceInstanceToSignalMapping

Class	ServiceInterface			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. Tags: atp.Status=draft atp.recommendedPackage=ServiceInterfaces			
Base	ARElement, AROObject, 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

Table A.97: 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			
Base	ARElement, AROObject, CollectableElement, Identifiable , MultilanguageReferrable, Packageable Element, Referrable , UploadablePackageElement			
Subclasses	DdisServiceInterfaceDeployment , SomeipServiceInterfaceDeployment , UserDefinedServiceInterface Deployment			
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.98: ServiceInterfaceDeployment

Class	ServiceInterfaceElementSecureComConfig			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	This element allows to secure the communication of the referenced ServiceInterface element. Tags: atp.ManifestKind=ServiceInstanceManifest 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.
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.
getterCall	ServiceField Deployment	0..1	ref	Reference to a field getter call message that is protected by a security protocol. Tags: atp.Status=draft





Class	ServiceInterfaceElementSecureComConfig			
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
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.99: ServiceInterfaceElementSecureComConfig

Class	<i>ServiceMethodDeployment</i> (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, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
Subclasses	<i>DdsMethodDeployment</i> , <i>SomeipMethodDeployment</i> , <i>UserDefinedMethodDeployment</i>			
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.100: 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
exactOrAnyMinorVersion	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





Enumeration	ServiceVersionAcceptanceKindEnum
minimumMinorVersion	Search for ANY minor version service instance and select only those service instances which have an equal or greater minor version than given in requiredMinorVersion. Tags:atp.EnumerationLiteralIndex=1

Table A.101: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
udpCollectionBufferTimeout	TimeValue	0..1	attr	Maximum time, an outgoing message (event, method call or method response) may be delayed, due to data collection.
udpCollectionTrigger	UdpCollectionTriggerEnum	0..1	attr	Defines whether the ServiceInterface element (event or method) contributes to the triggering of the udp data transmission if data collection is enabled.

Table A.102: 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	DataPrototypeInServiceInterfaceRef	*	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
networkRepresentation	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





Class	SomeipDataPrototypeTransformationProps			
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.103: SomeipDataPrototypeTransformationProps

Class	SomeipEventDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for an Event. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable , ServiceEventDeployment			
Attribute	Type	Mult.	Kind	Note
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.
maximumSegmentLength	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.
separationTime	TimeValue	0..1	attr	Sets the duration of the minimum time in seconds SOME/IP shall wait between the transmissions of segments.
serializer	SerializationTechnology Enum	0..1	attr	Defines which serialization technology shall be used.
transportProtocol	TransportLayerProtocol Enum	1	attr	This attribute defines over which Transport Layer Protocol this event is intended to be sent.

Table A.104: 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.ManifestKind=ServiceInstanceManifest 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





Class	SomeipEventGroup			
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.

Table A.105: 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.ManifestKind=ServiceInstanceManifest 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.106: SomeipEventProps

Class	SomeipFieldDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a Field. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i> , <i>ServiceFieldDeployment</i>			
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.107: SomeipFieldDeployment

Class	SomeipMethodDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a Method. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, Identifiable , MultilanguageReferrable , Referrable , ServiceMethodDeployment			
Attribute	Type	Mult.	Kind	Note
maximumSegmentLengthRequest	PositiveInteger	0..1	attr	<p>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.</p> <p>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.</p>
maximumSegmentLengthResponse	PositiveInteger	0..1	attr	<p>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.</p> <p>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.</p>
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.
separationTimeRequest	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.
separationTimeResponse	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.
transportProtocol	TransportLayerProtocol Enum	1	attr	This attribute defines over which Transport Layer Protocol this method is intended to be sent.

Table A.108: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
collectionProps	SomeipCollectionProps	0..1	aggr	<p>Collection of timing attributes configurable for a method that is provided or requested by a Service Instance.</p> <p>Tags:atp.Status=draft</p>





SomeipMethodProps				
Class				
method	SomeipMethodDeployment	0..1	ref	Reference to the method for which the SomeipMethod Props are applicable. Tags: atp.Status=draft

Table A.109: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
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
eventMulticastUdpPort	PositiveInteger	0..1	attr	UdpPort configuration that is used for Event communication in the IP-Multicast case. During SOME/IP Service Discovery: Send in the SD-SubscribeEventGroupAck Message to client (answer to SD-SubscribeEventGroup). Event: This is the destination-port where the server sends the multicast event messages if the multicastThreshold is exceeded.
ipv4MulticastIp Address	Ip4AddressString	0..1	attr	Multicast IPv4 Address that is transmitted in the Event GroupSubscribeAck message.
ipv6MulticastIp Address	Ip6AddressString	0..1	attr	Multicast IPv6 Address that is transmitted in the Event GroupSubscribeAck message.
multicastThreshold	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.
sdServerEventGroupTimingConfig	SomeipSdServerEventGroupTimingConfig	0..1	ref	Server Timing configuration settings that are EventGroup specific. Tags: atp.Status=draft

Table A.110: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	<i>ARObject, Referrable</i>			
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
sdClientEventGroupTimingConfig	SomeipSdClientEventGroupTimingConfig	1	ref	Client Timing configuration settings that are EventGroup specific. Tags: atp.Status=draft

Table A.111: 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.ManifestKind=ServiceInstanceManifest atp.recommendedPackage=SomeipSdTimingConfigs			
Base	<i>ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
requestResponseDelay	RequestResponseDelay	0..1	aggr	The Service Discovery shall delay answers to unicast messages triggered by multicast messages (e.g. Subscribe Eventgroup after Offer Service).
subscribeEventgroupRetryDelay	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).
subscribeEventgroupRetryMax	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 long 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.112: 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.ManifestKind=ServiceInstanceManifest atp.recommendedPackage=SomeipSdTimingConfigs			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
initialFind Behavior	InitialSdDelayConfig	0..1	aggr	Controls initial find behavior of clients.
serviceFind TimeToLive	PositiveInteger	1	attr	This attribute represents the ability to define the time in seconds the service find is valid.

Table A.113: 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.ManifestKind=ServiceInstanceManifest atp.recommendedPackage=SomeipSdTimingConfigs			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
initialOffer Behavior	InitialSdDelayConfig	0..1	aggr	Controls offer behavior of the server. Tags: atp.Status=draft
offerCyclicDelay	TimeValue	0..1	attr	Optional attribute to define cyclic offers. Cyclic offer is active, if the delay is set (in seconds).
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). Tags: atp.Status=draft
serviceOffer TimeToLive	PositiveInteger	1	attr	Defines the time in seconds the service offer is valid.

Table A.114: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInstanceToMachineMappings			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referrable</i> , <i>ServiceInstanceToMachineMapping</i> , UploadablePackageElement			





Class	SomeipServiceInstanceToMachineMapping			
Attribute	Type	Mult.	Kind	Note
tcpPort	PositiveInteger	0..1	attr	<p>TcpPort configuration that is used for Method and Event communication in IP-Unicast case.</p> <p>During SOME/IP Service Discovery: PortNumber that is sent in the SD-Offer Message to client (answer on SD-find) or clients (SD-offer).</p> <p>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).</p> <p>Event: This is the event source-port where the server sends the event messages to the subscribed clients in IP-Unicast case.</p>
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.
udpPort	PositiveInteger	0..1	attr	<p>UdpPort configuration that is used for Method and Event communication in IP-Unicast case.</p> <p>During SOME/IP Service Discovery: PortNumber that is sent in the SD-Offer Message to client (answer on SD-find) or clients (SD-offer).</p> <p>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).</p> <p>Event: This is the event source-port where the server sends the event messages to the subscribed clients in IP-Unicast case.</p>

Table A.115: SomeipServiceInstanceToMachineMapping

Class	SomeipServiceInterfaceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a ServiceInterface. Tags: atp.ManifestKind=ServiceInstanceManifest 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	<p>SOME/IP EventGroups that are defined within the SOME/IP ServiceClass.</p> <p>Tags:atp.Status=draft</p>
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.





Class	SomeipServiceInterfaceDeployment			
serviceInterfaceVersion	SomeipServiceVersion	1	aggr	The SOME/IP major and minor Version of the Service. Tags: atp.Status=draft

Table A.116: 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.ManifestKind=ServiceInstanceManifest atp.Status=draft			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
majorVersion	PositiveInteger	0..1	attr	Major Version of the ServiceInterface. Tags: xml.sequenceOffset=10
minorVersion	PositiveInteger	1	attr	Minor Version of the ServiceInterface. Tags: xml.sequenceOffset=20

Table A.117: 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 , CppImplementationData TypeContextTarget, Identifiable , MultilanguageReferrable, PackageableElement, Referrable			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.118: 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: ▽			



Class	<>atpVariation>> SwDataDefProps			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
additionalNativeTypeQualifier	NativeDeclarationString	0..1	attr	<p>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.</p> <p>Tags: xml.sequenceOffset=235</p>
annotation	Annotation	*	aggr	<p>This aggregation allows to add annotations (yellow pads ...) related to the current data object.</p> <p>Tags: xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false</p>
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	<p>This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system.</p> <p>Tags: xml.sequenceOffset=210</p>
displayPresentation	DisplayPresentationEnum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementationDataType	AbstractImplementationDataType	0..1	ref	<p>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</p> <ul style="list-style-type: none"> • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype





Class	<<atpVariation>> SwDataDefProps			
				<p style="text-align: right;">△</p> <ul style="list-style-type: none"> the target type of a pointer (see SwPointerTargetProps), if it does not refer to a base type directly the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly the data type of an SwServiceArg, if it does not refer to a base type directly <p>Tags:xml.sequenceOffset=215</p>
invalidValue	ValueSpecification	0..1	aggr	<p>Optional value to express invalidity of the actual data element.</p> <p>Tags:xml.sequenceOffset=255</p>
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.
				<p>Tags:xml.sequenceOffset=30</p>
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memory AllocationKeywordPolicy of the referenced SwAddr Method.
				<p>Tags:xml.sequenceOffset=33</p>
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable.
				<p>Tags:xml.sequenceOffset=60</p>
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object.
				<p>Tags:xml.sequenceOffset=70</p>
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.
				<p>Tags:xml.sequenceOffset=90</p>
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process.
				<p>Tags:</p> <p>xml.sequenceOffset=170</p> <p>xml.typeElement=false</p>
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).
				<p>Tags:xml.sequenceOffset=200</p>
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.
				<p>Tags:</p> <p>xml.sequenceOffset=220</p> <p>xml.typeElement=false</p>





Class	<>atpVariation>> SwDataDefProps			
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. Tags: xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	<p>The purpose of this element is to describe the requested quantization of data objects early on in the design process.</p> <p>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).</p> <p>In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution.</p> <p>The resolution is specified in the physical domain according to the property "unit".</p> <p>Tags:xml.sequenceOffset=240</p>
swInterpolation Method	Identifier	0..1	attr	<p>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.</p> <p>Tags:xml.sequenceOffset=250</p>
swIsVirtual	Boolean	0..1	attr	<p>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 .</p> <p>Tags:xml.sequenceOffset=260</p>
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	<p>Specifies that the containing data object is a pointer to another data object.</p> <p>Tags:xml.sequenceOffset=280</p>
swRecord Layout	SwRecordLayout	0..1	ref	<p>Record layout for this data object.</p> <p>Tags:xml.sequenceOffset=290</p>
swRefresh Timing	MultidimensionalTime	0..1	aggr	<p>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.</p> <p>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.</p> <p>Tags:xml.sequenceOffset=300</p>
swTextProps	SwTextProps	0..1	aggr	<p>the specific properties if the data object is a text object.</p> <p>Tags:xml.sequenceOffset=120</p>
swValueBlock Size	Numerical	0..1	attr	<p>This represents the size of a Value Block</p> <p>Stereotypes: atpVariation</p> <p>Tags:</p> <p>vh.latestBindingTime=preCompileTime xml.sequenceOffset=80</p>





Class	<>atpVariation>> SwDataDefProps			
swValueBlock SizeMult (ordered)	Numerical	*	attr	<p>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.</p> <p>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.</p> <p>For one-dimensional value blocks the attribute swValueBlockSize shall be used and this attribute shall not exist.</p> <p>Stereotypes: atpVariation Tags:vh.latestBindingTime=preCompileTime</p>
unit	Unit	0..1	ref	<p>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.</p> <p>Tags:xml.sequenceOffset=350</p>
valueAxisDataType	ApplicationPrimitive DataType	0..1	ref	<p>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.</p> <p>Tags:xml.sequenceOffset=355</p>

Table A.119: 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
arraySizeSemantics	ArraySizeSemantics Enum	1	attr	<p>This attribute controls the semantics of the arraysize for the array representing the string in an Implementation DataType.</p> <p>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.</p>
baseType	SwBaseType	0..1	ref	<p>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 ApplicationDataType.</p> <p>Tags:xml.sequenceOffset=30</p>
swFillCharacter	Integer	0..1	attr	<p>Filler character for text parameter to pad up to the maximum length swMaxTextSize.</p> <p>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</p>





Class	SwTextProps			
	<p style="text-align: right;">△</p> <p>filler character and 0 (dec) represents an end of string as filler character.</p> <p>The usage of the fill character depends on the arraySize Semantics.</p> <p>Tags:xml.sequenceOffset=40</p>			
swMaxTextSize	Integer	1	attr	<p>Specifies the maximum text size in characters. Note the size in bytes depends on the encoding in the corresponding baseType.</p> <p>Stereotypes: atpVariation</p> <p>Tags: vh.latestBindingTime=preCompileTime xml.sequenceOffset=20</p>

Table A.120: SwTextProps

Class	SymbolProps			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to contribute a part of a namespace.			
Base	ARObject, <i>ImplementationProps, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
-	-	-	-	-

Table A.121: SymbolProps

Class	TlsSecureComProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	Configuration of the Transport Layer Security protocol (TLS).			
	<p>Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft</p>			
Base	ARObject, <i>Identifiable, MultilanguageReferrable, Referrable, SecureComProps</i>			
Attribute	Type	Mult.	Kind	Note
keyExchange	CryptoServicePrimitive	*	ref	<p>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.</p> <p>Tags:atp.Status=draft</p>
tlsCipherSuite	TlsCryptoCipherSuite	*	aggr	<p>Collection of supported cipher suites that are used to negotiate the security settings for a network connection defined by the ServiceInstanceToMachineMapping.</p> <p>Tags:atp.Status=draft</p>

Table A.122: TlsSecureComProps

Class	TlvDataIdDefinition			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::SerializationProperties			
Note	This meta-class represents the ability to define the tlvDataId. Tags: atp.Status=draft			
Base	ARObject			
Attribute	Type	Mult.	Kind	Note
id	PositiveInteger	1	attr	This attribute represents the definition of the value of the TlvDataId
tlvArgument	ArgumentDataPrototype	0..1	ref	This reference assigns a tlvDataId to a given argument of a ClientServerOperation. Tags: atp.Status=draft
tlvRecord Element	ApplicationRecord Element	0..1	ref	This reference associates the definition of a TLV data id with a given ApplicationRecordElement. Tags: atp.Status=draft
tlvSubElement	CpplImplementation DataTypeElement	0..1	ref	This reference associates the definition of a TLV data id with a given CpplImplementationDataTypeElement. Stereotypes: atpSplitable Tags: atp.Splitkey=tlvSubElement atp.Status=draft

Table A.123: 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			
Base	ARObject, <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>Referrable</i>			
Attribute	Type	Mult.	Kind	Note
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





Class	TransformationPropsToServiceInterfaceElementMapping			
transformationProps	TransformationProps	0..1	ref	This represents the reference to the applicable Serialization properties. Tags: atp.Status=draft

Table A.124: TransformationPropsToServiceInterfaceElementMapping

Class	TransformationPropsToServiceInterfaceElementMappingSet			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::SerializationProperties			
Note	Collection of TransformationPropsToServiceInterfaceElementMappings. Tags: atp.Status=draft atp.recommendedPackage=TransformationPropsToServiceInterfaceMappingSets			
Base	ARElement, AROObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, Packageable Element, <i>Referable</i>			
Attribute	Type	Mult.	Kind	Note
mapping	TransformationPropsToServiceInterfaceElementMapping	*	aggr	Mapping that assigns serialization properties to elements of a ServiceInterface. Tags: atp.Status=draft

Table A.125: TransformationPropsToServiceInterfaceElementMappingSet

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			
udp	User datagram protocol Tags: atp.EnumerationLiteralIndex=0			

Table A.126: TransportLayerProtocolEnum

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			
never	ServiceInterface element will be buffered and will not trigger the transmission of the data. Tags: atp.EnumerationLiteralIndex=1			

Table A.127: UdpCollectionTriggerEnum

Class	UserDefinedServiceInterfaceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	UserDefined configuration settings for a ServiceInterface. Tags: atp.ManifestKind=ServiceInstanceManifest atp.Status=draft atp.recommendedPackage=ServiceInterfaceDeployments			
Base	<i>ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, Referrable, ServiceInterfaceDeployment, UploadablePackageElement</i>			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table A.128: 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	<i>ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, Multilanguage Referrable, Referrable</i>			
Attribute	Type	Mult.	Kind	Note
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table A.129: VariableDataPrototype

B History of Specification Items

B.1 Constraint and Specification Item History of this document according to AUTOSAR Release 17-10

B.1.1 Added Traceables in 17-10

Number	Heading
[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





Number	Heading
[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 <code>baseTypeSize</code> of 16
[SWS_CM_00428]	Element specification typed by String Data Type with <code>baseTypeSize</code> 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]	





Number	Heading
[SWS_CM_10278]	
[SWS_CM_10279]	
[SWS_CM_10280]	
[SWS_CM_10281]	
[SWS_CM_10282]	
[SWS_CM_10283]	
[SWS_CM_10284]	
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10286]	Encoding mismatch in input configurations
[SWS_CM_10287]	Conditions for sending of a SOME/IP event message
[SWS_CM_10288]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10289]	Source of a SOME/IP event message
[SWS_CM_10290]	Destination of a SOME/IP event message
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10293]	Identifying the right event
[SWS_CM_10294]	Deserializing the payload
[SWS_CM_10295]	Store the received event data
[SWS_CM_10296]	Invoke receive handler
[SWS_CM_10297]	Conditions for sending of a SOME/IP request message
[SWS_CM_10298]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10299]	Source of a SOME/IP request message
[SWS_CM_10300]	Destination of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10303]	Identifying the right method
[SWS_CM_10304]	Deserializing the payload
[SWS_CM_10305]	Store the received method data
[SWS_CM_10306]	Invoke the method - event driven
[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





Number	Heading
[SWS_CM_10315]	Discarding orphaned responses
[SWS_CM_10316]	Deserializing the payload - response messages
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10318]	Invoke the notification function
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10320]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10321]	Source of a SOME/IP event message
[SWS_CM_10322]	Destination of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10324]	Checks for a received SOME/IP event message
[SWS_CM_10325]	Identifying the right event
[SWS_CM_10326]	Deserializing the payload
[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_10351]	Service application errors





Number	Heading
[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 messages
[SWS_CM_10361]	
[SWS_CM_10362]	Raising checked exceptions for application errors
[SWS_CM_10370]	Data Type definitions for Application Errors in Common header file
[SWS_CM_10371]	Context of thrown checked exceptions
[SWS_CM_11262]	
[SWS_CM_11263]	
[SWS_CM_90101]	Secure channel creation
[SWS_CM_90102]	Using secure channels
[SWS_CM_90103]	TLS secure channel for methods using reliable transport
[SWS_CM_90104]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90107]	TLS secure channel for fields
[SWS_CM_90108]	SecOC secure channel for methods
[SWS_CM_90109]	SecOC secure channel for events
[SWS_CM_90110]	SecOC secure channel for fields
[SWS_CM_90401]	
[SWS_CM_90402]	
[SWS_CM_90403]	
[SWS_CM_90404]	
[SWS_CM_90405]	
[SWS_CM_90406]	
[SWS_CM_90407]	
[SWS_CM_90408]	
[SWS_CM_90409]	
[SWS_CM_90410]	
[SWS_CM_90411]	
[SWS_CM_90412]	
[SWS_CM_90413]	



△

Number	Heading
[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 B.1: Added Traceables in 17-10

B.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
[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

▽

△

Number	Heading
[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 <code>baseTypeSize</code> of 8
[SWS_CM_00409]	Associative Map Data Type
[SWS_CM_00420]	Element specification typed by String Data Type with <code>baseTypeSize</code> 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 B.2: Changed Traceables in 17-10

B.1.3 Deleted Traceables in 17-10

Number	Heading
[SWS_CM_01003]	Inclusion protection

Table B.3: Deleted Traceables in 17-10

B.2 Constraint and Specification Item History of this document according to AUTOSAR Release 18-03

B.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
[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.





Number	Heading
[SWS_CM_01047]	Every record element shall have a <code>wire</code> 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 <code>tag</code> assigned when the optionality is used for at least one record element inside the struct.
[SWS_CM_01049]	The <code>tlvDataIds</code> shall be synchronized between the interacting proxy and skeleton instances.
[SWS_CM_01050]	<code>Variant</code> Class Template
[SWS_CM_01051]	<code>Variant</code> default constructor
[SWS_CM_01052]	<code>Variant</code> move constructor
[SWS_CM_01053]	<code>Variant</code> copy constructor
[SWS_CM_01054]	<code>Variant</code> destructor
[SWS_CM_01055]	<code>Variant</code> move assignment operator
[SWS_CM_01056]	<code>Variant</code> default copy assignment operator
[SWS_CM_01057]	<code>Variant</code> function to return the zero-based index of the alternative
[SWS_CM_01058]	<code>Variant</code> 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 <code>CompuScales</code> with non-point range
[SWS_CM_10377]	Sending SOME/IP SubscribeEventgroup messages - initial
[SWS_CM_10378]	Sending SOME/IP StopSubscribeEventgroup messages
[SWS_CM_10379]	Silently discarding SOME/IP event messages for unsubscribed events
[SWS_CM_10380]	Silently discarding SOME/IP event messages for unsubscribed events
[SWS_CM_10381]	Sending SOME/IP SubscribeEventgroup messages - renewal
[SWS_CM_10382]	Calling stop find service for already stopped finds
[SWS_CM_10384]	Change of Service Interface Deployment
[SWS_CM_10385]	Change of Service Instance Deployment
[SWS_CM_10386]	Change of Network Configuration
[SWS_CM_10387]	Data accumulation for UDP data transmission
[SWS_CM_10388]	Enabling of data accumulation for UDP data transmission
[SWS_CM_10389]	Configuration of a data accumulation on a <code>ProvidedServiceInstance</code> for transmission over UDP
[SWS_CM_10390]	Configuration of a data accumulation on a <code>RequiredSomeipServiceInstance</code> for transmission over UDP





Number	Heading
[SWS_CM_11000]	
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11002]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_11003]	Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface
[SWS_CM_11004]	Adding Service and Service Instance IDs to the DDS Domain Participant's USER_DATA QoS Policy
[SWS_CM_11005]	Mapping of StopOfferService method
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11007]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11008]	Creating a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11010]	Mapping of StartFindService method
[SWS_CM_11011]	Defining a DDS BuiltinParticipantListener
[SWS_CM_11012]	Binding a BuiltinParticipantListener to a DDS DomainParticipant
[SWS_CM_11013]	Mapping of StopFindService method
[SWS_CM_11014]	Unbinding a BuiltinParticipantListener from a DDS DomainParticipant
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11016]	DDS Topic datatype definition
[SWS_CM_11017]	Mapping of Send method
[SWS_CM_11018]	Mapping of Subscribe method
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11020]	Defining a DDS DataReaderListener
[SWS_CM_11021]	Mapping of Unsubscribe method
[SWS_CM_11022]	Mapping of GetSubscriptionState method
[SWS_CM_11023]	Mapping of Update method
[SWS_CM_11024]	Mapping of GetCachedSamples method
[SWS_CM_11025]	Mapping of SetReceiveHandler method
[SWS_CM_11026]	Mapping of UnsetReceiveHandler method
[SWS_CM_11027]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11028]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11041]	
[SWS_CM_11042]	
[SWS_CM_11043]	
[SWS_CM_11044]	Serialization of Strings of <code>baseTypeSize 8</code>
[SWS_CM_11045]	Serialization of Strings of <code>baseTypeSize 16</code>



△

Number	Heading
[SWS_CM_11046]	Serialization of <code>ImplementationDataType</code> of category VECTOR
[SWS_CM_11047]	Serialization of <code>ImplementationDataType</code> 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 B.4: Added Traceables in 18-03

B.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
[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

▽



Number	Heading
[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 <code>baseTypeSize</code> of 8
[SWS_CM_00407]	Vector Data Type with one dimension
[SWS_CM_00408]	Vector Data Type with more than one dimension
[SWS_CM_00409]	Associative Map Data Type
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00411]	Avoid Data Type redeclaration
[SWS_CM_00413]	Element specification typed by Base Type
[SWS_CM_00414]	Element specification typed by Implementation Data Type
[SWS_CM_00415]	Element specification typed by Array





Number	Heading
[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 <code>baseTypeSize</code> 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 <code>Enumeration Data Types</code>
[SWS_CM_00427]	String Data Type with <code>baseTypeSize</code> of 16
[SWS_CM_00428]	Element specification typed by String Data Type with <code>baseTypeSize</code> 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]	

▽



Number	Heading
[SWS_CM_10281]	
[SWS_CM_10282]	
[SWS_CM_10283]	
[SWS_CM_10284]	
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10286]	Encoding mismatch in input configurations
[SWS_CM_10287]	Conditions for sending of a SOME/IP event message
[SWS_CM_10288]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10289]	Source of a SOME/IP event message
[SWS_CM_10290]	Destination of a SOME/IP event message
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10293]	Identifying the right event
[SWS_CM_10294]	Deserializing the payload
[SWS_CM_10295]	Store the received event data
[SWS_CM_10296]	Invoke receive handler
[SWS_CM_10297]	Conditions for sending of a SOME/IP request message
[SWS_CM_10298]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10299]	Source of a SOME/IP request message
[SWS_CM_10300]	Destination of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10303]	Identifying the right method
[SWS_CM_10304]	Deserializing the payload
[SWS_CM_10305]	Store the received method data
[SWS_CM_10306]	Invoke the method - event driven
[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





Number	Heading
[SWS_CM_10318]	Invoke the notification function
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10320]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10321]	Source of a SOME/IP event message
[SWS_CM_10322]	Destination of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10324]	Checks for a received SOME/IP event message
[SWS_CM_10325]	Identifying the right event
[SWS_CM_10326]	Deserializing the payload
[SWS_CM_10327]	Store the received event data
[SWS_CM_10328]	Invoke receive handler
[SWS_CM_10329]	Conditions for sending of a SOME/IP request message
[SWS_CM_10330]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10331]	Source of a SOME/IP request message
[SWS_CM_10332]	Destination of a SOME/IP request message
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10335]	Identifying the right method
[SWS_CM_10336]	Deserializing the payload
[SWS_CM_10337]	Store the received method data
[SWS_CM_10338]	Invoke the registered set/get handlers - event driven
[SWS_CM_10339]	Invoke the registered set/get handlers - polling
[SWS_CM_10340]	Conditions for sending of a SOME/IP response message
[SWS_CM_10341]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_10342]	Source of a SOME/IP response message
[SWS_CM_10343]	Destination of a SOME/IP response message
[SWS_CM_10344]	Content of the SOME/IP response message
[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



△

Number	Heading
[SWS_CM_10359]	Deserializing the payload - error response messages
[SWS_CM_10361]	
[SWS_CM_11262]	
[SWS_CM_11263]	
[SWS_CM_90103]	TLS secure channel for methods using reliable transport
[SWS_CM_90104]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90401]	
[SWS_CM_90402]	
[SWS_CM_90403]	
[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 B.5: Changed Traceables in 18-03

B.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
[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 B.6: Deleted Traceables in 18-03

B.3 Constraint and Specification Item History of this document according to AUTOSAR Release 18-10

B.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 <code>arraySize</code> of an <code>CppImplementationDataType</code> with category VECTOR
[SWS_CM_00502]	<code>CustomCppMethodType</code> of category ARRAY
[SWS_CM_00503]	<code>StdCppMethodType</code> of category VECTOR with one dimension defined with an <code>Allocator</code>
[SWS_CM_00504]	Supported Primitive Cpp Implementation Data Types
[SWS_CM_00505]	<code>StdCppMethodType</code> with category ASSOCIATIVE_MAP defined with an <code>Allocator</code>
[SWS_CM_00506]	<code>CustomCppMethodType</code> of category ASSOCIATIVE_MAP





Number	Heading
[SWS_CM_00507]	CustomCppType of category VECTOR
[SWS_CM_00508]	CustomCppType of category VARIANT
[SWS_CM_00509]	StdCppType 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_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 destroying and initializing the contained value
[SWS_CM_01069]	Variant function to create a new value in-place, in an existing Variant object by destroying 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



△

Number	Heading
[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 ScaleLinearAndTexttables 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 ScaleLinearAndTexttables 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

▽



Number	Heading
[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
[SWS_CM_11049]	DDS serialization of <code>CppImplementationDataType</code> of category ASSOCIATIVE_MAP
[SWS_CM_11050]	DDS serialization of <code>CppImplementationDataType</code> 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





Number	Heading
[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
[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





Number	Heading
[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 B.7: Added Traceables in 18-10

B.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





Number	Heading
[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
[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]	<code>StdCppImplementationDataType</code> of category <code>ARRAY</code> with one dimension
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00405]	Structure Data Type
[SWS_CM_00406]	<code>StdCppImplementationDataType</code> with the category <code>STRING</code>
[SWS_CM_00407]	<code>StdCppImplementationDataType</code> of category <code>VECTOR</code> with one dimension defined without an <code>Allocator</code>
[SWS_CM_00408]	Vector Data Type with more than one dimension
[SWS_CM_00409]	<code>StdCppImplementationDataType</code> with category <code>ASSOCIATIVE_MAP</code> defined without an <code>Allocator</code>
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00411]	Avoid Data Type redeclaration
[SWS_CM_00414]	Element specification typed by <code>CppImplementationDataType</code>
[SWS_CM_00421]	Provide data type definitions
[SWS_CM_00423]	Data Type Mapping





Number	Heading
[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_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 tlvDataIds between the interacting proxy and skeleton instances.
[SWS_CM_01050]	Variant Class Template
[SWS_CM_01054]	Variant converting constructor
[SWS_CM_01055]	Variant explicit converting constructor with specified alternative
[SWS_CM_01056]	Variant explicit converting constructor with specified alternative and initializer list
[SWS_CM_01057]	Variant explicit converting constructor with alternative specified by index
[SWS_CM_01058]	Variant explicit converting constructor with alternative specified by index and initializer list
[SWS_CM_10017]	
[SWS_CM_10036]	
[SWS_CM_10042]	
[SWS_CM_10059]	
[SWS_CM_10070]	
[SWS_CM_10234]	
[SWS_CM_10235]	
[SWS_CM_10242]	Model representation of UTF-8 Strings
[SWS_CM_10245]	Serialization of strings
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10253]	
[SWS_CM_10262]	Insertion of an associative map length field
[SWS_CM_10265]	Serialization of associative map elements
[SWS_CM_10285]	Responsibility of proper string encoding





Number	Heading
[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
[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





Number	Heading
[SWS_CM_10390]	Configuration of a data accumulation on a <code>RequiredSomeipServiceInstance</code> 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
[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 <code>StdCppImplementationDataType</code> of category <code>VALUE</code>
[SWS_CM_11042]	DDS serialization of enumeration data types
[SWS_CM_11043]	DDS serialization of <code>StdCppImplementationDataType</code> of category <code>STRUCTURE</code>
[SWS_CM_11044]	DDS serialization of <code>StdCppImplementationDataType</code> of category <code>STRING</code> with string <code>shortName</code>
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS



△

Number	Heading
[SWS_CM_11047]	DDS serialization of <code>CppImplementationDataType</code> of category VECTOR
[SWS_CM_11048]	DDS serialization of <code>CppImplementationDataType</code> of category ARRAY
[SWS_CM_90001]	Restrictions on executing methods
[SWS_CM_90101]	Secure UDP and TCP channel creation for TLS, DTLS and SecOC
[SWS_CM_90102]	Using secure TLS, DTLS and SecOC channels
[SWS_CM_90103]	TLS secure channel for methods using reliable transport
[SWS_CM_90104]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90108]	SecOC secure channel for methods using reliable transport
[SWS_CM_90109]	SecOC secure channel for events using reliable transport
[SWS_CM_90110]	SecOC secure channel for fields
[SWS_CM_90401]	
[SWS_CM_90404]	
[SWS_CM_90420]	E2ECheckStatus of a sample
[SWS_CM_90421]	ara::com:E2E_state_machine::E2Echeckstatus
[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 B.8: Changed Traceables in 18-10

B.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

▽

△

Number	Heading
[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 <code>baseTypeSize</code> of 8
[SWS_CM_00422]	Reject data type definitions
[SWS_CM_00427]	String Data Type with <code>baseTypeSize</code> of 16
[SWS_CM_00428]	Element specification typed by String Data Type with <code>baseTypeSize</code> of 16
[SWS_CM_00448]	Element specification typed by Variant
[SWS_CM_00451]	Namespace specification for an ImplementationDataType of category VECTOR
[SWS_CM_01033]	Optional Class Template
[SWS_CM_01034]	Optional default constructor
[SWS_CM_01035]	Optional move constructor
[SWS_CM_01036]	Optional copy constructor
[SWS_CM_01037]	Optional destructor
[SWS_CM_01038]	Optional move assignment operator
[SWS_CM_01039]	Optional default copy assignment operator
[SWS_CM_01040]	Optional function to get contained value
[SWS_CM_01041]	Optional function to check availability of contained value
[SWS_CM_01042]	Optional bool operator

▽

△

Number	Heading
[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 messages
[SWS_CM_11045]	Serialization of Strings of <code>baseTypeSize 16</code>
[SWS_CM_90432]	Functionality of Sample Pointer

Table B.9: Deleted Traceables in 18-10

B.4 Constraint and Specification Item History of this document according to AUTOSAR Release 19-03

B.4.1 Added Traceables in 19-03

none

B.4.2 Changed Traceables in 19-03

none

B.4.3 Deleted Traceables in 19-03

none

B.5 Constraint and Specification Item History of this document according to AUTOSAR Release 19-11

B.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 <code>invalidValue</code>
[SWS_CM_10454]	
[SWS_CM_10455]	
[SWS_CM_10456]	
[SWS_CM_10457]	
[SWS_CM_10458]	Handling of an ServiceInterface that does not contain any events, methods, or fields
[SWS_CM_10459]	
[SWS_CM_10460]	
[SWS_CM_10461]	
[SWS_CM_10462]	
[SWS_CM_10463]	





Number	Heading
[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]	
[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





Number	Heading
[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
[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





Number	Heading
[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
[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



△

Number	Heading
[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]	<code>ServiceInstanceToSignalMapping</code> 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 B.10: Added Traceables in R19-11

B.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]	<code>StdCppImplementationDataType</code> of category <code>ARRAY</code> with one dimension

▽

△

Number	Heading
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00405]	Structure Data Type
[SWS_CM_00406]	StdCppImplementationDataType with the category STRING
[SWS_CM_00407]	StdCppImplementationDataType of category VECTOR with one dimension defined without an Allocator
[SWS_CM_00409]	StdCppImplementationDataType with category ASSOCIATIVE_MAP defined without an Allocator
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00414]	Element specification typed by CppImplementationDataType
[SWS_CM_00424]	Enumeration Data Type
[SWS_CM_00425]	Definition of enumerators
[SWS_CM_00449]	Variant Data Type
[SWS_CM_00502]	CustomCppType of category ARRAY
[SWS_CM_00503]	StdCppType of category VECTOR with one dimension defined with an Allocator
[SWS_CM_00504]	Supported Primitive Cpp Implementation Data Types
[SWS_CM_00505]	StdCppType with category ASSOCIATIVE_MAP defined with an Allocator
[SWS_CM_00506]	CustomCppType of category ASSOCIATIVE_MAP
[SWS_CM_00507]	CustomCppType of category VECTOR
[SWS_CM_00508]	CustomCppType of category VARIANT
[SWS_CM_00509]	StdCppType 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

▽

△

Number	Heading
[SWS_CM_01020]	Folder structure
[SWS_CM_01031]	Service fields namespace
[SWS_CM_01032]	Accessing optional record elements inside a Structure Cpp Implementation Data Type that are serialized with the Tag-Length-Value principle.
[SWS_CM_01046]	Definition of tlvDataIdDefinition
[SWS_CM_01050]	Variant Class Template
[SWS_CM_01051]	Variant default constructor
[SWS_CM_01052]	Variant move constructor
[SWS_CM_01053]	Variant copy constructor
[SWS_CM_01054]	Variant converting constructor
[SWS_CM_01055]	Variant explicit converting constructor with specified alternative
[SWS_CM_01056]	Variant explicit converting constructor with specified alternative and initializer list
[SWS_CM_01057]	Variant explicit converting constructor with alternative specified by index
[SWS_CM_01058]	Variant explicit converting constructor with alternative specified by index and initializer list
[SWS_CM_01059]	Variant destructor
[SWS_CM_01060]	Variant move assignment operator
[SWS_CM_01061]	Variant default copy assignment operator
[SWS_CM_01062]	Variant converting assignment operator
[SWS_CM_01063]	Variant function to return the zero-based index of the alternative
[SWS_CM_01064]	Variant function to check if the Variant is in invalid state
[SWS_CM_01065]	Variant function to swap two Variants
[SWS_CM_01066]	Variant function to create a new value in-place, in an existing Variant object
[SWS_CM_01067]	Variant function to create a new value in-place, in an existing Variant object using an initializer list
[SWS_CM_01068]	Variant function to create a new value in-place, in an existing Variant object by destroying and initializing the contained value
[SWS_CM_01069]	Variant function to create a new value in-place, in an existing Variant object by destroying 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

▽



Number	Heading
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10295]	Providing the received event data
[SWS_CM_10299]	Source of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10308]	Conditions for sending of a SOME/IP response message
[SWS_CM_10310]	Source of a SOME/IP response message
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10313]	Checks for a received SOME/IP response message
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10327]	Providing the received event data
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10358]	Identifying the right application error in a message with Message Type set to RESPONSE (0x80)
[SWS_CM_10362]	Raising checked errors for application errors
[SWS_CM_10372]	Inclusion of Implementation Types header files
[SWS_CM_10373]	Implementation Types header files existence
[SWS_CM_10375]	Implementation Types header file namespace
[SWS_CM_10383]	GetHandle function to return the proxy instance creation handle
[SWS_CM_10384]	Change of Service Interface Deployment
[SWS_CM_10385]	Change of Service Instance Deployment
[SWS_CM_10392]	ScaleLinearAndTexttable Class Template
[SWS_CM_10393]	ScaleLinearAndTexttable static assertion
[SWS_CM_10394]	ScaleLinearAndTexttable underlying type deduction
[SWS_CM_10395]	ScaleLinearAndTexttable default constructor
[SWS_CM_10396]	ScaleLinearAndTexttable copy constructor
[SWS_CM_10397]	ScaleLinearAndTexttable constructor with enum class argument
[SWS_CM_10398]	ScaleLinearAndTexttable constructor with underlying type argument
[SWS_CM_10399]	ScaleLinearAndTexttable copy assignment operator
[SWS_CM_10400]	ScaleLinearAndTexttable assignment operator with enum class argument
[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 <code>ScaleLinearAndTexttable</code> objects
[SWS_CM_10404]	Equal to operators between <code>ScaleLinearAndTexttable</code> and an underlying type
[SWS_CM_10405]	Equal to operators between <code>ScaleLinearAndTexttables</code> and an enum class
[SWS_CM_10406]	Not equal to operator between two <code>ScaleLinearAndTexttable</code> objects
[SWS_CM_10407]	Not equal to operators between <code>ScaleLinearAndTexttable</code> and an underlying type
[SWS_CM_10408]	Not equal to operators between <code>ScaleLinearAndTexttables</code> and an enum class
[SWS_CM_10409]	Scale Linear And Textable type definition
[SWS_CM_10414]	Initiate a method call
[SWS_CM_10428]	payload representing application error
[SWS_CM_10430]	Handling invalid messages with Message Type set to RESPONSE (0x80)
[SWS_CM_10431]	Mapping of ara::core::ErrorCode
[SWS_CM_10432]	
[SWS_CM_10433]	Declaration of Construction Token
[SWS_CM_10434]	Creation of a Construction Token
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10438]	Exception-less creation of service proxy
[SWS_CM_10450]	InstanceSpecifier check during the creation of service skeleton
[SWS_CM_10451]	InstanceIdIdentifierContainer 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

▽



Number	Heading
[SWS_CM_11041]	DDS serialization of <code>StdCppImplementationDataType</code> of category VALUE
[SWS_CM_11043]	DDS serialization of <code>StdCppImplementationDataType</code> of category STRUCTURE
[SWS_CM_11044]	DDS serialization of <code>StdCppImplementationDataType</code> of category STRING with string <code>shortName</code>
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS
[SWS_CM_11047]	DDS serialization of <code>CppImplementationDataType</code> of category VECTOR
[SWS_CM_11048]	DDS serialization of <code>CppImplementationDataType</code> of category ARRAY
[SWS_CM_11102]	DDS Service Reply Topic data type definition
[SWS_CM_11132]	Mapping of Update method
[SWS_CM_11133]	Mapping of Subscribe method
[SWS_CM_11134]	Creating a DDS DataReader for field subscription
[SWS_CM_11136]	Mapping of Unsubscribe method
[SWS_CM_11138]	Mapping of GetNewSamples method
[SWS_CM_11139]	Mapping of GetFreeSampleCount method
[SWS_CM_11145]	DDS Service Request Topic data type definition for Field getter and setter operations
[SWS_CM_11146]	DDS Service Reply Topic data type definition for Field getter and setter operations
[SWS_CM_11264]	Definition general ara::com errors
[SWS_CM_11265]	Use of general ara::com errors
[SWS_CM_11266]	Definition of Application Errors
[SWS_CM_90001]	Restrictions on executing methods
[SWS_CM_90002]	Restrictions on sending events
[SWS_CM_90003]	Restrictions on receiving events
[SWS_CM_90005]	Restrictions on offering services
[SWS_CM_90006]	Restrictions on using services
[SWS_CM_90113]	Behavior of a ServiceSkeleton over TLS before successful completion of the handshake
[SWS_CM_90114]	Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake
[SWS_CM_90118]	Transport of Service communication over an IPsec security association
[SWS_CM_90401]	
[SWS_CM_90402]	
[SWS_CM_90403]	
[SWS_CM_90404]	
[SWS_CM_90405]	
[SWS_CM_90406]	



△

Number	Heading
[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
[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 B.11: Changed Traceables in R19-11

B.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

▽

△

Number	Heading
[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]	
[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 B.12: Deleted Traceables in R19-11