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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	17-10

Document Change History			
Date	**Release**	**Changed by**	**Description**
2017-10-27	17-10	AUTOSAR Release Management	• Introduction of Fields • Introduction of E2E protected communication • Introduction of TLV • Improved specification of SOME/IP functional behavior • Minor changes and bugfixes
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

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

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

The Communication Management provides a 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
E2E	End-to-end communication protection

Terms:	Description:
Service Binding	Act of connecting a Service Requester to a concrete Service Instance of a Service Provider.
Multi-Binding	Multi-Binding describes setups having multiple connections implemented by different technical transport layers and protocol between different instances of a single proxy or skeleton class, e.g.: • A proxy class uses different transport/IPC to communicate with different skeleton instances. • Different proxy instances for the same skeleton instance uses different transport/IPC to communicate with this instance: The skeleton instance supports multiple transport mechanisms to get contacted.

3 Related documentation

3.1 Input documents

[1] Explanation of ara::com API
AUTOSAR_EXP_ARAComAPI

[2] Glossary
AUTOSAR_TR_Glossary

[3] SOME/IP Protocol Specification
AUTOSAR_PRS_SOMEIPProtocol

[4] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[5] Requirements on Communication Management
AUTOSAR_RS_CommunicationManagement

[6] Requirements on E2E
AUTOSAR_RS_E2E

[7] E2E Protocol Specification
AUTOSAR_PRS_E2EProtocol

[8] SOME/IP Service Discovery Protocol Specification
AUTOSAR_PRS_SOMEIPServiceDiscoveryProtocol

[9] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes

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

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

[12] Methodology for Adaptive Platform
AUTOSAR_TR_AdaptiveMethodology

[13] General Specification of Adaptive Platform
AUTOSAR_SWS_General

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

[15] ISO/IEC TS 19571:2016, Programming Languages – Technical specification for C++ extensions for concurrency
http://www.iso.org

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

[17] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[18] Guidelines for the use of the C++14 language in critical and safety-related systems
AUTOSAR_RS_CPP14Guidelines

3.2 Related standards and norms

See chapter 3.1.

3.3 Related specification

See chapter 3.1.

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:

- **SubscriptionState**
  The state of a service subscription will be accessible by using the function GetSubscriptionState(). For preliminary information how this should look like from the point of view of applications, please refer to chapter *6.1.3.2 Monitoring Event Subscription* of *Explanation of ara::com API* [1].

The following functionality is treated as critical but not worked out currently:

- **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 E2E communication protection is works only for events which are polled and which are transmitted at least once per fault tolerant time interval. This means, it requires:

- Periodic invocation of the method `Update` in a polling mode
- Periodic or mixed-periodic invocation of the method `Send`

In case `Update` or `Send` are not invoked periodically, then some communication failure modes are not detected (loss, delay and possibly also repetition). In this case, if E2E is used, then additional measures need to be taken at application level to address those non-detected failure modes.

`EndToEndTransformationComSpecProps` are not supported.

The following limitations regarding optionality introduced with the Tag-Length-Value serialization principle described in [3] and [4] apply:

- **Optional method arguments**
  The Specification does not support the existence of optional method arguments.
- **Definition of `wire type`s 4 to 7 for Complex Data Types**
  The definition on sender side of which `wire type` should be used for Complex Data Types is implementation defined.

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other functional clusters

There are currently no dependencies to other functional clusters.

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Communication Management document [5] and links to the fulfillment of these.

Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_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_01014] [SWS_CM_01016] [SWS_CM_01017] [SWS_CM_01019] [SWS_CM_01020] [SWS_CM_10370]
[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_10351]
[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_00201] [SWS_CM_00203] [SWS_CM_00302]
[RS_CM_00102]	Communication Management shall provide an interface to find services	[SWS_CM_00004] [SWS_CM_00121] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00124] [SWS_CM_00125] [SWS_CM_00131] [SWS_CM_00202] [SWS_CM_00209] [SWS_CM_00303] [SWS_CM_00304] [SWS_CM_00305] [SWS_CM_00312] [SWS_CM_10352] [SWS_CM_10353]
[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]
[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]
[RS_CM_00105]	Communication Management shall provide an interface to stop offering services	[SWS_CM_00111] [SWS_CM_00204]
[RS_CM_00200]	The Communication Management shall transform Fully Qualified Service IDs to communication protocol specific Service IDs	[SWS_CM_01010]

Requirement	Description	Satisfied by
[RS_CM_00201]	Communication Management shall provide an API to send events to other applications	[SWS_CM_00003] [SWS_CM_00161] [SWS_CM_00162] [SWS_CM_00163] [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_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_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10234] [SWS_CM_10242] [SWS_CM_10243] [SWS_CM_10245] [SWS_CM_10247] [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] [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_10286] [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
[RS_CM_00202]	Communication Management shall provide an API to the application to poll received events	[SWS_CM_00171] [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] [SWS_CM_00300] [SWS_CM_00306] [SWS_CM_00307] [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_10169] [SWS_CM_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10234] [SWS_CM_10242] [SWS_CM_10243] [SWS_CM_10245] [SWS_CM_10247] [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] [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_10286] [SWS_CM_10295] [SWS_CM_10327] [SWS_CM_10361]
[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_00300] [SWS_CM_00306] [SWS_CM_00307] [SWS_CM_00309] [SWS_CM_10296] [SWS_CM_10328]
[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_10000]

Requirement	Description	Satisfied by
[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_01033] [SWS_CM_01034] [SWS_CM_01035] [SWS_CM_01036] [SWS_CM_01037] [SWS_CM_01038] [SWS_CM_01039] [SWS_CM_01040] [SWS_CM_01041] [SWS_CM_01042] [SWS_CM_01043] [SWS_CM_01044] [SWS_CM_01045] [SWS_CM_01046] [SWS_CM_01047] [SWS_CM_01048] [SWS_CM_01049] [SWS_CM_10000]
[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_00262] [SWS_CM_00263] [SWS_CM_00264] [SWS_CM_00265] [SWS_CM_00301] [SWS_CM_00400] [SWS_CM_00401] [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_00413] [SWS_CM_00414] [SWS_CM_00415] [SWS_CM_00416] [SWS_CM_00418] [SWS_CM_00419] [SWS_CM_00420] [SWS_CM_00421] [SWS_CM_00422] [SWS_CM_00423] [SWS_CM_00424] [SWS_CM_00425] [SWS_CM_00426] [SWS_CM_00427] [SWS_CM_00428] [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_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10234] [SWS_CM_10242] [SWS_CM_10243] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10252] [SWS_CM_10253] [SWS_CM_10256] [SWS_CM_10257] [SWS_CM_10258]

Requirement	Description	Satisfied by
		[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_10286] [SWS_CM_10354] [SWS_CM_10355] [SWS_CM_10356] [SWS_CM_10361] [SWS_CM_10362] [SWS_CM_10371]
[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_10305] [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_10335] [SWS_CM_10336] [SWS_CM_10337] [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_10359] [SWS_CM_10362] [SWS_CM_10371]
[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_10305] [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]

Requirement	Description	Satisfied by
		[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_10337] [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_10359] [SWS_CM_10362] [SWS_CM_10371]
[RS_CM_00214]	Communication Management shall provide an interface to query the result of an asynchronously called service method	[SWS_CM_00193] [SWS_CM_00320] [SWS_CM_00321] [SWS_CM_00322] [SWS_CM_00323] [SWS_CM_00324] [SWS_CM_00325] [SWS_CM_00326] [SWS_CM_00327] [SWS_CM_00328] [SWS_CM_00329] [SWS_CM_00330] [SWS_CM_00332] [SWS_CM_00340] [SWS_CM_00341] [SWS_CM_00342] [SWS_CM_00343] [SWS_CM_00344] [SWS_CM_00345] [SWS_CM_00346] [SWS_CM_00347] [SWS_CM_00348] [SWS_CM_10362] [SWS_CM_10371]
[RS_CM_00215]	Communication Management shall trigger the application on completion of an asynchronously called service method	[SWS_CM_00197] [SWS_CM_00321] [SWS_CM_00331] [SWS_CM_00340] [SWS_CM_00341] [SWS_CM_00342] [SWS_CM_00343] [SWS_CM_00344] [SWS_CM_00345] [SWS_CM_00346] [SWS_CM_00347] [SWS_CM_00348] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10349] [SWS_CM_10350]
[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]
[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]
[RS_CM_00219]	Communication Management shall provide an interface which aggregates methods to send an event 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]

Requirement	Description	Satisfied by
[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]
[RS_CM_00223]	Communication Management shall protect the transmission of data using E2E protocol, hidden behind the event API.	[SWS_CM_90433]
[RS_E2E_08534]	E2E Protocol shall provide error information for the detected communication failure	[SWS_CM_90411] [SWS_CM_90413] [SWS_CM_90416] [SWS_CM_90417] [SWS_CM_90418] [SWS_CM_90419] [SWS_CM_90420] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90423] [SWS_CM_90424] [SWS_CM_90431]
[RS_E2E_08540]	E2E protocol shall support protected periodic/mixed periodic communication	[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_90415] [SWS_CM_90416] [SWS_CM_90417] [SWS_CM_90430] [SWS_CM_90433]
[RS_SEC_03018]	Capabilities of an Adaptive Application shall be authentically linked to the manifest	[SWS_CM_90001]
[RS_SEC_03019]	No description	[SWS_CM_90001] [SWS_CM_90002] [SWS_CM_90003] [SWS_CM_90004]
[RS_SEC_04001]	Secure communication shall be performed through 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]
[RS_SEC_04003]	The assignment of communication to secure channels shall be defined	[SWS_CM_90102]
[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]
[RS_SOMEIPSD_00015]	SOME/IP Service Discovery Protocol shall support to subscribe to events	[SWS_CM_00206]
[RS_SOMEIPSD_00016]	SOME/IP Service Discovery Protocol shall support to deny subscriptions	[SWS_CM_00208]
[RS_SOMEIPSD_00028]	SOME/IP Service Discovery shall support configurable timings	[SWS_CM_00201] [SWS_CM_00209]

Requirement	Description	Satisfied by
[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]
[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]
[RS_SOMEIP_00005]	SOME/IP protocol shall support different strategies for event communication	[SWS_CM_10034] [SWS_CM_10287] [SWS_CM_10319]
[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_10305] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10314]
[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_10305] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10308] [SWS_CM_10309] [SWS_CM_10310] [SWS_CM_10311] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10316] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10329] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10332] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10336] [SWS_CM_10337] [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_10359]
[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_10359]

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_10337] [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]
[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]
[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]
[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]
[RS_SOMEIP_00017]	SOME/IP protocol shall support grouping events into eventgroups	[SWS_CM_10287] [SWS_CM_10319]
[RS_SOMEIP_00018]	SOME/IP protocol shall support grouping fields in eventgroups	[SWS_CM_10319]
[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]
[RS_SOMEIP_00021]	SOME/IP protocol shall identify RPC methods of services using unique identifiers	[SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346]
[RS_SOMEIP_00022]	SOME/IP protocol shall identify events of services using unique identifiers	[SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10293] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325]
[RS_SOMEIP_00025]	SOME/IP protocol shall support the identification of callers of an RPC using unique identifiers	[SWS_CM_10301] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10333] [SWS_CM_10344] [SWS_CM_10345]
[RS_SOMEIP_00026]	SOME/IP protocol shall define the endianness of header and payload	[SWS_CM_10013] [SWS_CM_10172]
[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_10359]

Requirement	Description	Satisfied by
[RS_SOMEIP_00041]	SOME/IP protocol shall provide support of multiple versions of the protocol	[SWS_CM_10291] [SWS_CM_10301] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10323] [SWS_CM_10333] [SWS_CM_10344] [SWS_CM_10345]
[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]
[RS_SOMEIP_00050]	SOME/IP protocol shall support serialization of extensible data structs	[SWS_CM_01032] [SWS_CM_01033] [SWS_CM_01034] [SWS_CM_01035] [SWS_CM_01036] [SWS_CM_01037] [SWS_CM_01038] [SWS_CM_01039] [SWS_CM_01040] [SWS_CM_01041] [SWS_CM_01042] [SWS_CM_01043] [SWS_CM_01044] [SWS_CM_01045] [SWS_CM_01046] [SWS_CM_01047] [SWS_CM_01048] [SWS_CM_01049]
[SRS_Xfrm_00101]	No description	[SWS_CM_11262] [SWS_CM_11263]
[S_CM_00212]	No description	[SWS_CM_10334]

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

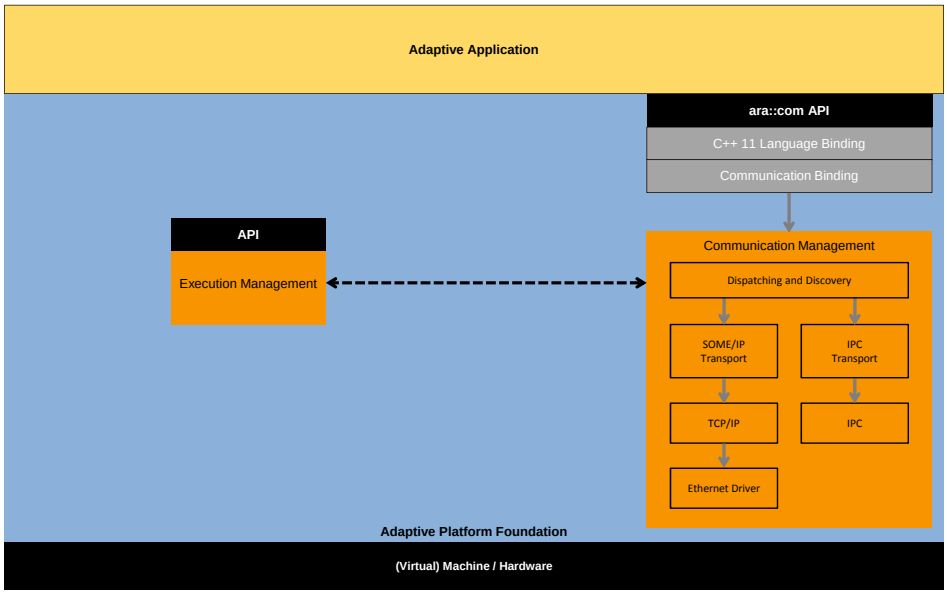

Figure 7.1: Technical Architecture of Communication Management

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

- Public Interface: Part of the Adaptive AUTOSAR API and specified in the SWS. This is the standardized ara::com API.
- Protected Interface: 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)
- Private Interface: Interaction 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 both static and dynamic communication:

  - Full static configuration, service discovery not needed at all as the server knows all clients and clients know the server
  - 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 and 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 is sub-classing 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.

- 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 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) is the main communication pattern for Adaptive AUTOSAR Applications. It allows establishing communication paths both at design- and run-time, so it can be used to build up both static communication with known numbers of participants and dynamic communication with unknown number of participants. Figure 7.2 shows the basic operation principle of Service-Oriented Communication.

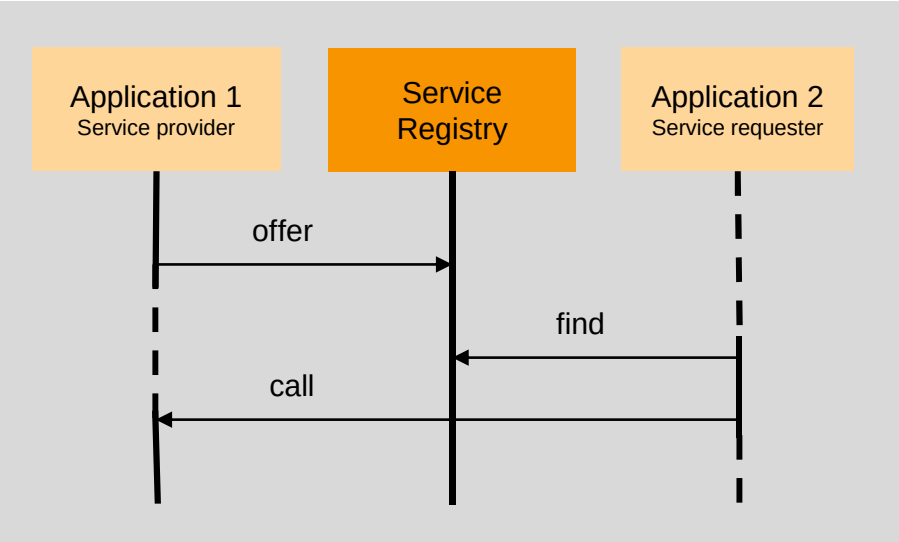

Figure 7.2: Service-Oriented Communication

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.

The Communication Management software using Service-Oriented Communication will not achieve hard real time requirements, as the implementation will behave like a virtual ethernet including latencies of communication. This behavior must be respected with the design of the overall ECU and SW system.

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.

7.2 End-to-end communication protection

This section specifies the integration of `ara::e2e` within `ara::com` for processing periodic events, that are polled by the `Subscriber`.

[SWS_CM_90402] ⌈ An e2e-protected event shall have its options configured in `End2EndEventProtectionProps` and `E2EProfileConfiguration`. ⌋ *(RS_E2E_08540)*

[SWS_CM_90433] ⌈ The E2E functions mentioned in this section - `E2EProtect` and `E2ECheck` - shall comply with the E2E protection protocol as defined in as specified in [6] and [7]. ⌋*(RS_E2E_08540, RS_CM_00223)*

7.2.1 Publisher

[SWS_CM_90401] ⌈ For e2e-protected events, E2E protection shall be performed within the context of `Send`, by means of `Send` invoking `E2ECheck`. ⌋*(RS_E2E_08540)*

Figure 7.3 shows an overview of the interaction of components involved during the E2E protection.

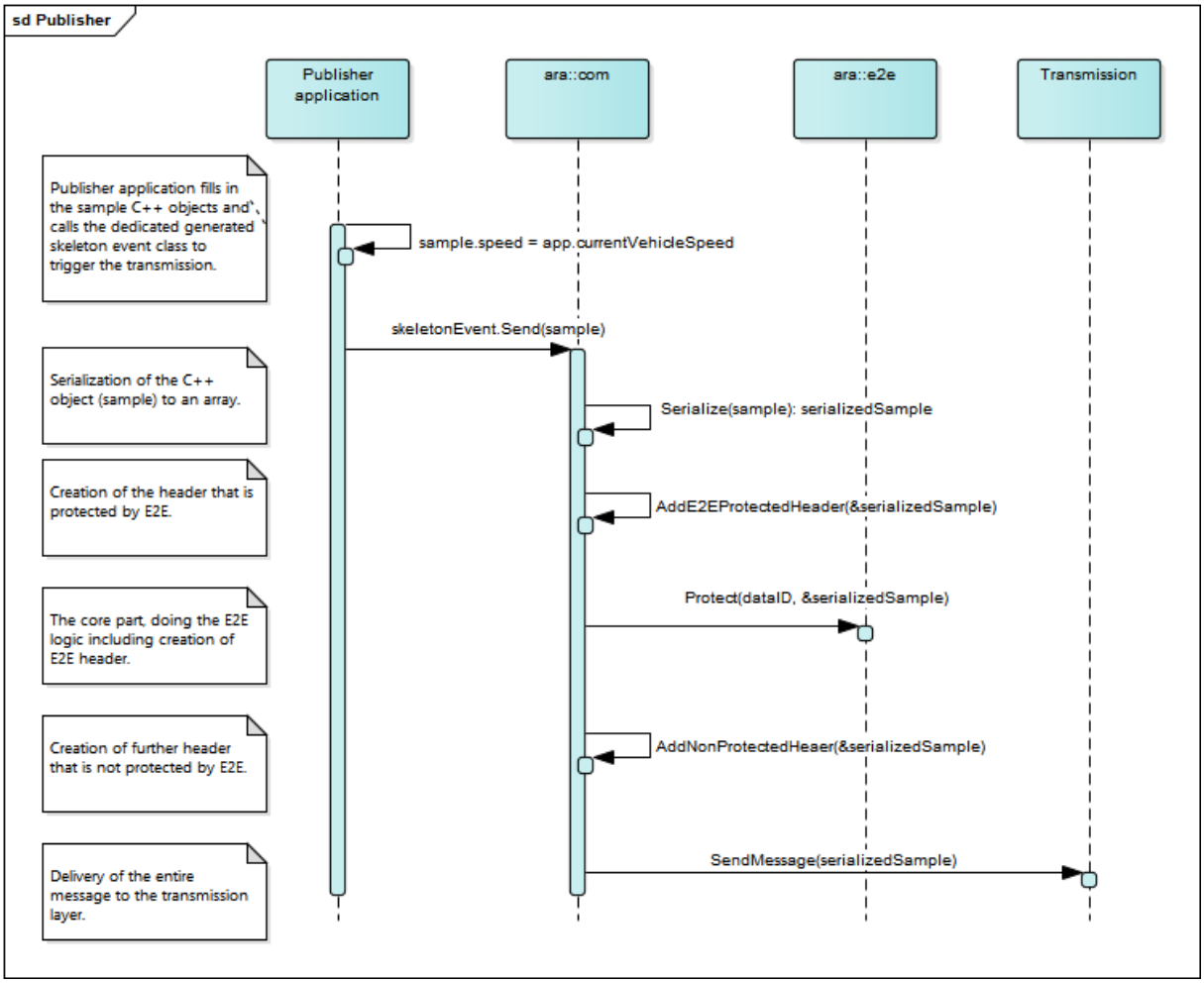

Figure 7.3: E2E Publisher

[SWS_CM_90430] ⌈ For e2e-protected events, `Send` shall serialize the `sample` according to the agreed serialization protocol, resulting with `sample`. ⌋*(RS_E2E_08540)*

[SWS_CM_90403] ⌈ For e2e-protected events, `Send` shall determine `dataID`, based on `Service ID`, `Instance ID` and `Event ID` of this `Event` instance. ⌋ *(RS_E2E_08540)*

[SWS_CM_90404] ⌈ For e2e-protected events, `Send` shall provide the `serialized-Sample` to `E2ECheck`, where `serializedSample` is made of (1) the header that is part of e2e protection and (2) the serialized data. ⌋*(RS_E2E_08540)*

[SWS_CM_90405] ⌈ 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)*

7.2.2 Subscriber - Update

[SWS_CM_90406] ⌈ For e2e-protected events, E2E `Check` shall be performed within the context of `Update`. ⌋*(RS_E2E_08540)*

Figure 7.4 shows an overview of the interaction of components involved during the E2E check.

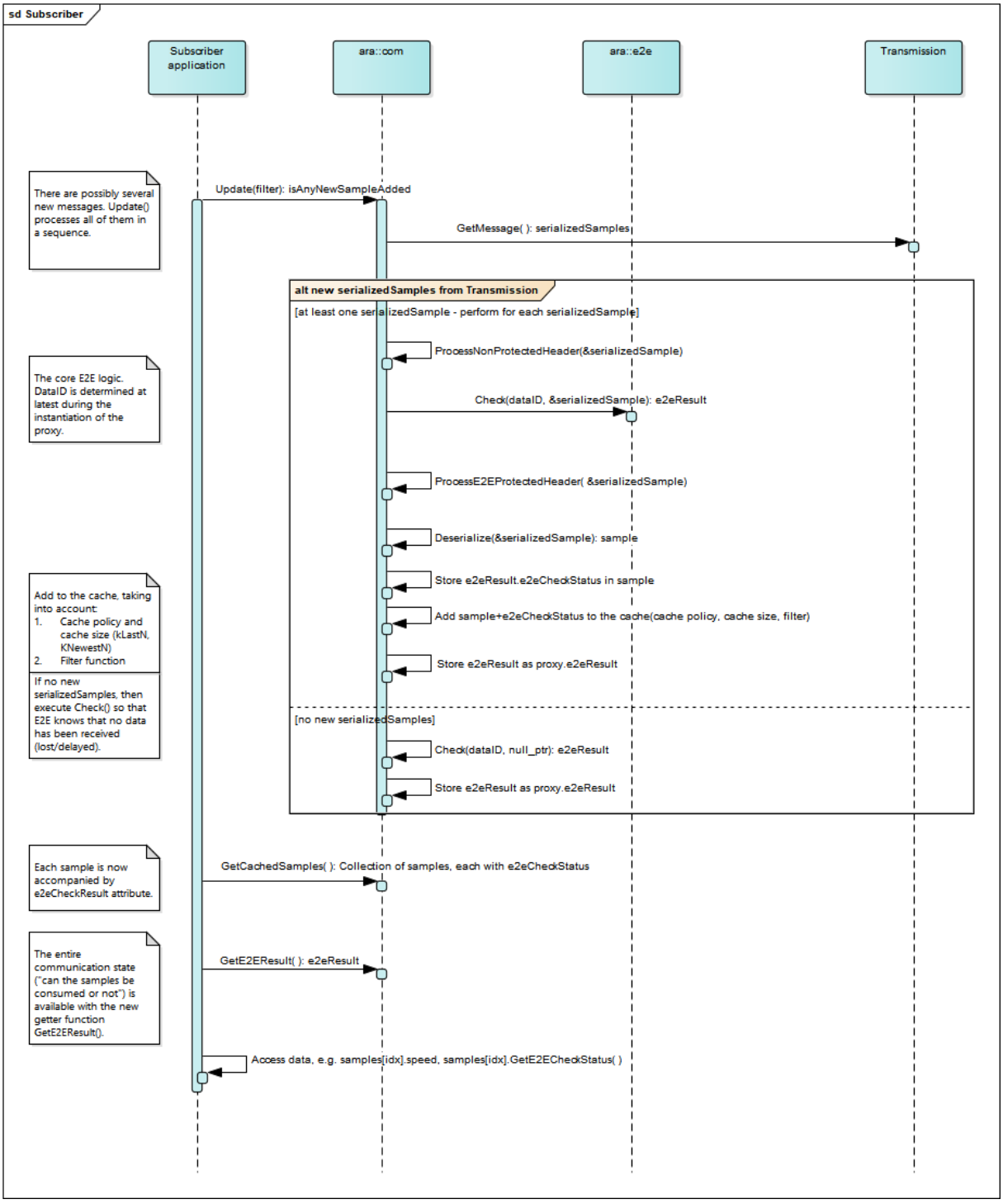

Figure 7.4: E2E Subscriber

[SWS_CM_90407] ⌈ For e2e-protected events, `Update` shall first get the collection of all `SerializedSample`s that appeared after the last triggering of this `Update` function. ⌋*(RS_E2E_08540)*

7.2.2.1 Case 1 - there are one or more serialized samples

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] ⌈ For the given e2e-protected SerializedSample, Update shall process the non-e2e protected header (if any) of the serializedSample. ⌋ *(RS_E2E_08540)*

[SWS_CM_90409] ⌈ Update shall determine the DataID based on Service ID, Service Instance ID, Event ID of this Event instance. ⌋*(RS_E2E_08540)*

[SWS_CM_90410] ⌈ For the given e2e-protected SerializedSample, Update shall invoke the E2ECheck, providing to it dataID and serializedSample. ⌋ *(RS_E2E_08540)*

[SWS_CM_90411] ⌈ In return, for the given e2e-protected SerializedSample, E2ECheck shall provide E2EResult containing E2EState and E2ECheckStatus. ⌋*(RS_E2E_08540, RS_E2E_08534)*

[SWS_CM_90412] ⌈ For the given e2e-protected SerializedSample, Update shall deserialize it, resulting with deserialized sample. ⌋*(RS_E2E_08540)*

[SWS_CM_90413] ⌈ For the given e2e-protected SerializedSample, Update shall store the pair sample and e2eCheckStatus in the application cache and it shall update/overwrite event.e2eState with e2eResult.e2eState. ⌋*(RS_E2E_08540, RS_E2E_08534)*

7.2.2.2 Case 2 - there are no serialized samples

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

[SWS_CM_90414] ⌈ In case no e2e-protected SerializedSamples are received, Update shall determine the DataID based on Service ID, Service Instance ID, Event ID of this Event instance. ⌋*(RS_E2E_08540)*

[SWS_CM_90415] ⌈ In case no e2e-protected SerializedSamples are received, Update shall invoke the E2ECheck, providing to it dataID and null sample. ⌋ *(RS_E2E_08540)*

[SWS_CM_90416] ⌈ In case no e2e-protected SerializedSamples are received, in return, E2ECheck shall provide E2EResult containing E2EState and E2ECheckStatus. ⌋*(RS_E2E_08540, RS_E2E_08534)*

[SWS_CM_90417] ⌈ In case no e2e-protected SerializedSamples are received, Update shall store the pair sample and e2eCheckStatus in the application cache and it shall update/overwrite event.e2eState with e2eResult.e2eState. ⌋ *(RS_E2E_08540, RS_E2E_08534)*

7.2.3 Subscriber - GetCachedSamples

[SWS_CM_90418] ⌈ `GetCachedSamples` shall provide a collection of smart pointers to pairs made of (`sample` and `e2eCheckStatus`), where the collection contains the samples determined/provided in the most recent invocation of `Update` according to the selected cache policy. ⌋*(RS_E2E_08534)*

7.2.4 Subscriber - Access to E2E information

[SWS_CM_90419] ⌈ Each `sample` shall have a getter function `GetE2ECheckStatus` allowing to access e2eCheckStatus of each `Sample`. ⌋*(RS_E2E_08534)*

[SWS_CM_90431] ⌈ Each `Event` shall have a getter function `GetE2EState` allowing to access e2eState that was determined by the last run of `E2ECheck` function invoked during the last `Update` of the `Event`. ⌋*(RS_E2E_08534)*

7.3 Network binding

The following chapters describe the requirements according to specific bus protocol bindings. In the current version, only SOME/IP is supported.

7.3.1 SOME/IP Network binding

[SWS_CM_10000] ⌈ The SOME/IP network binding shall implement the SOME/IP Protocol and the SOME/IP Service Discovery Protocol defined in [3] and [8]. ⌋ *(RS_CM_00204, RS_CM_00205)*

[SWS_CM_10013] ⌈ All headers shall be encoded in network byte order Big Endian (MostSignificantByteFirst) [RFC 791]. ⌋*(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_SOMEIP_00026)*

7.3.1.1 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_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).

[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 of the SOME/IP service discovery protocol. ⌋*(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).

[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 serviceInterface. 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 SomeipServiceInterfaceVersion element that is aggregated by the RequiredSomeipServiceInstance in the role requiredServiceVersion defines the majorVersion. If the majorVersion is set to "ANY" then 0xFF shall be used.
- Minor Version of the RequiredSomeipServiceInstance that is searched shall be derived from the Manifest where the SomeipServiceInterfaceVersion element that is aggregated by the RequiredSomeipServiceInstance in the role requiredServiceVersion defines the minorVersion. If the minorVersion is set to "ANY" then 0xFFFF FFFF shall be used.
- TTL shall be derived from the Manifest where the SomeipSdClientServiceInstanceConfig element that is aggregated by the RequiredSomeipServi-

ceInstance 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 SomeipSdClientServiceInstanceConfig element that is aggregated by the RequiredSomeipServiceInstance in the role sdClientConfig. The content of the Configuration Option shall be derived from the key/value pairs defined in each capabilityRecord.

⌋(*RS_CM_00102*, *RS_SOMEIPSD_00006*)

[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 serviceInterface.
- Major Version of the SomeipServiceInterfaceDeployment that is offered shall be derived from the Manifest where the SomeipServiceInterfaceVersion 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 SomeipServiceInterfaceVersion element that is aggregated by the SomeipServiceInterfaceDeployment in the role serviceInterfaceVersion defines the minorVersion .
- TTL shall be derived from the Manifest where the SomeipSdServerServiceInstanceConfig element that is aggregated 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 theIpv4Configuration 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 theIpv6Configuration element.

- The Transport Layer Protocol used in the IPv4 Endpoint option and/or IPv6 Endpoint option shall be derived from the Manifest where the SomeipServiceInstanceToMachineMapping element that maps the ProvidedSomeipServiceInstance to an EthernetCommunicationConnector of a Machine defines the TP and PortNumber.
  - UDP shall be used if SomeipServiceInstanceToMachineMapping.udpPort is configured.
  - TCP shall be used if SomeipServiceInstanceToMachineMapping.tcpPort is configured.
- Configuration Option shall be used in the offer message if at least one capabilityRecord is defined for the ProvidedSomeipServiceInstance in the aggregated SomeipSdServerServiceInstanceConfig. The content of the Configuration Option shall be derived from the key/value pairs defined in each capabilityRecord.

⌋(*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.
- Eventgroup ID 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_00105*, *RS_SOMEIPSD_00006*)

[SWS_CM_00205] 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.
- Minor Version shall be derived from the offer message.
- Eventgroup ID shall be derived from Manifest where the RequiredSomeipServiceInstance element aggregates the SomeipRequiredEventGroup in the role requiredEventGroup. The SomeipRequiredEventGroup contains the eventGroup reference to the SomeipEventGroup where the eventGroupId is defined.
- TTL shall be derived from Manifest where the RequiredSomeipServiceInstance element aggregates the SomeipRequiredEventGroup in the role requiredEventGroup. The SomeipRequiredEventGroup aggregates the sdClientEventTimingConfig where the timeToLive is defined.
- IPv4 Endpoint Option shall be sent if the offer message contains an IPv4 Endpoint Option. In this case the IPv4 Address sent in the IPv4 Endpoint Option of the SubscribeEventgroup message is configured in the Manifest where the RequiredSomeipServiceInstance element is mapped with the ServiceInstanceToMachineMapping to an EthernetCommunicationConnector of a Machine. The EthernetCommunicationConnector refers to a NetworkEndpoint in the role unicastNetworkEndpoint where an IPv4 Address is configured in theIpv4Configuration element.
- IPv6 Endpoint Option shall be sent if the offer message contains an IPv6 Endpoint Option. In this case the IPv6 Address sent in the IPv6 Endpoint Option of the SubscribeEventgroup message is configured in the Manifest where the RequiredSomeipServiceInstance element is mapped with the ServiceInstanceToMachineMapping to an EthernetCommunicationConnector of a Machine. The EthernetCommunicationConnector refers to a NetworkEndpoint in the role unicastNetworkEndpoint where an IPv6 Address is configured in theIpv6Configuration element.
- The Transport Layer Protocol used in the IPv4 Endpoint option and/or IPv6 Endpoint option shall be derived from the Manifest where the SomeipEventGroup points either to SomeipEventDeployments where the transportProtocol is set to udp or to tcp. The SomeipServiceInstanceToMachineMapping element that maps the RequiredSomeipServiceInstance to an EthernetCommunicationConnector of a Machine defines the TP and PortNumber.
  - 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.
  - 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.

⌋(*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).
- ServiceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- InstanceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- Major Version shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- Minor 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 shall be derived from the Manifest if a multicastThreshold with a value greater 0 is defined for the SomeipProvidedEventGroup and a ipv4MulticastIpAddress is defined in the SomeipServiceInstanceToMachineMapping that maps the ProvidedSomeipServiceInstance that aggregates the SomeipProvidedEventGroup to an EthernetCommunicationConnector of a Machine.
- IPv6 Multicast Option shall shall be derived from the Manifest if a multicastThreshold with a value greater 0 is defined for the SomeipProvidedEventGroup and a ipv6MulticastIpAddress is defined in the SomeipServiceInstanceToMachineMapping that maps the ProvidedSomeipServiceInstance that aggregates the SomeipProvidedEventGroup to an EthernetCommunicationConnector of a Machine.
- 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 is mapped with the SomeipServiceInstanceToMachineMapping to an EthernetCommunicationConnector of a Machine. The SomeipServiceInstanceToMachineMapping defines the eventMulticastUdpPort.

⌋(*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).
- ServiceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- InstanceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- Major Version shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- Minor 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_SOMEIPSD_00016*, *RS_SOMEIPSD_00006*)

[SWS_CM_00207] 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.
- Minor 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_00104*, *RS_SOMEIPSD_00006*)

7.3.1.2 Handling Events

[SWS_CM_10287] Conditions for sending of a SOME/IP event message ⌈ The sending of a SOME/IP event message shall be initiated by invoking the `Send` method of the respective `Event` class (see [SWS_CM_00161]) 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 cancelled the subscripton 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_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017)*

[SWS_CM_10288] Transport protocol for sending of a SOME/IP event message ⌈ The SOME/IP event message shall be transmitted using UDP if the threshold defined by the `multicastThreshold` attribute of the `SomeipProvidedEventGroup` that is aggregated by the `ProvidedSomeipServiceInstance` in the role `eventGroup` in the Manifest has been reached (see [PRS_SOMEIPSD_00134]). The SOME/IP event message shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.eventDeployment.transportProtocol` in the Manifest if this threshold has not been reached (see [PRS_SOMEIPSD_00802]). ⌋*(RS_CM_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_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 mes-

sage, the one matching the selected transport protocol (see [SWS_CM_10289]) shall be used. ⌋*(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 (see section 7.3.1.5) 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 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 section 7.3.1.5.

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

[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 an Unchecked Exception shall be raised. ⌋ *(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_00201, RS_SOMEIP_00004, RS_SOMEIP_00022)*

[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 section 7.3.1.5. ⌋*(RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00028)*

[SWS_CM_10295] Store the received event data ⌈ The deserialized payload containing the event data shall be stored for rertrieval via the Update, GetCachedSamples, and Cleanup methods (see [SWS_CM_00171]) of the respective Event class for the event determined according to [SWS_CM_10293]. ⌋*(RS_CM_00202, 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_00203, RS_SOMEIP_00004)*

7.3.1.3 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 initiated 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_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_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00010)*

[SWS_CM_10299] Source of a SOME/IP request message ⌈ The SOME/IP request message shall use the unicast IP address defined in the Manifest by the `Ipv4Configuration`/`Ipv6Configuration` attribute of the `NetworkEndpoint` that is referenced (in role `unicastNetworkEndpoint`) by the `EthernetCommunicationConnector` of a `Machine` which in turn is mapped to the `RequiredSomeipServiceInstance` by means of a `SomeipServiceInstanceToMachineMapping` as source address for the transmission. The `udpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_10298]) is UDP. The `tcpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_10298]) is TCP. ⌋ *(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_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 (see section 7.3.1.5) in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) shall be set to a value that uniquely identifies the client within a `Machine`. - This may be achived by dynamically generating unique client IDs upon construction of the `ServiceProxy`.
- The Session ID (see [PRS_SOMEIP_00703]) shall be set to 0x0001 for the first call of a particular method by a given client and shall be incremented by 1 after each call performed by this client for the respective method (see [PRS_SOMEIP_00533]). Once the Session ID reaches 0xFFFF, it shall wrap around and start with 0x0001 again (see [PRS_SOMEIP_00521]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `REQUEST_NO_RETURN` (0x01) in case the `ClientServerOperation` referenced by `methodDeployment.method` contains a `fireAndForget` attribute which is set to `true`. The Message Type shall be set to `REQUEST` (0x00) otherwise.
- The Return Code (see [PRS_SOMEIP_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 `ArgumentDataPrototype`s of the `ClientServerOperation` which are not referenced by any of the `ClientServerOperation`'s possible `ApplicationError`s in role `errorContext` with `direction` set to `in` and `inout` serialized according to their order) according to section 7.3.1.5.

⌋(*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*)

[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 an Unchecked Exception shall be raised. ⌋(*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_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 section 7.3.1.5. ⌋(*RS_CM_00212*, *RS_CM_00213*, *RS_SOMEIP_00006*, *RS_SOMEIP_00007*, *RS_SOMEIP_00028*)

[SWS_CM_10305] Store the received method data ⌈ 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 stored for later processing (see [SWS_CM_10307]). ⌋(*RS_CM_00212*, *RS_CM_00213*, *RS_SOMEIP_00006*, *RS_SOMEIP_00007*)

[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_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_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 initiated upon the return (either via a normal return from the method or via the throwing one of the possible `ApplicationError`s referenced by the `ClientServerOperation` in the role `possibleError`) 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_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_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 `udpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_10309]) is UDP. The `tcpPort` shall be used as source port for the transmission in case the selected transport protocol (see [SWS_CM_10309]) is TCP. ⌋ *(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_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 (see section 7.3.1.5) 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 raised one of the possible ApplicationErrors referenced by the ClientServerOperation in role possibleError[1]. The Message Type shall be set to RESPONSE (0x80) otherwise.
- The Return Code (see [PRS_SOMEIP_00040]) shall be filled with the value of ApplicationError.errorCode increased by 0x1F (see [PRS_SOMEIP_00191]) in case the ClientServerOperation raised one of the possible ApplicationErrors referenced by the ClientServerOperation in role possibleError. The Return Code shall be set to E_OK (0x00) otherwise.
- The Payload shall contain the serialized payload according to section 7.3.1.5. – In case of a raised ApplicationError, the ArgumentDataPrototypes referenced by this ApplicationError in role errorContext shall be serialized according to their order[2]. The ArgumentDataPrototypes of the ClientServerOperation which are not referenced by any of the ClientServerOperation's possible ApplicationErrors in role errorContext with direction set to inout and out shall be serialized according to their order otherwise.

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

[2]Note that this is in fact an incompatibility with the AUTOSAR classic platform (i.e., in cases where an AUTOSAR adaptive platform server operates with an AUTOSAR classic platform client) which defines that all ArgumentDataPrototypes of the ClientServerOperation with direction set to inout and out shall be serialized according to their order in that case. – 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.

⌋(*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*)

[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 `SomeipMethodDeployment`s of the `SomeipServiceInterfaceDeployment`.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches `SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion`.
- Verify that the Client ID (see [PRS_SOMEIP_00702]) matches the client from the corresponding SOME/IP request message (see [SWS_CM_10301]).
- The Session ID (see [PRS_SOMEIP_00703]) matches the client from the corresponding SOME/IP request message (see [SWS_CM_10301]).

If any of the above checks fails the received SOME/IP response message shall be discarded and an `Unchecked Exception` shall be raised. ⌋(*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 `SomeipMethodDeployment`s of the `SomeipServiceInterfaceDeployment`, the right method shall be identified. ⌋(*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 cancelled according to [SWS_CM_00194] in the mean time, the received SOME/IP response message shall be silently discarded and any further processing shall be omitted. ⌋(*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)[3] w.r.t. the further processing according to [SWS_CM_10316], [SWS_CM_10359], and [SWS_CM_10317]. ⌋ *(RS_SOMEIP_00008)*

[SWS_CM_10316] Deserializing the payload - response mesages ⌈ Based on the method determined according to [SWS_CM_10314] the Payload of the response message shall be deserialized according to section 7.3.1.5. – Therefore the ArgumentDataPrototypes which are not referenced by any of the method's possible ApplicationErrors in role errorContext with direction set to inout and out shall be deserialized according to their order. ⌋*(RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00028)*

[SWS_CM_10358] Identifying the right application error ⌈ If the Return Code see [PRS_SOMEIP_00040]) contains a value larger than 0x1F the corresponding value of the ApplicationError.errorCode attribute shall be determined by subtracting 0x1F from the Return Code value. Using this computed ApplicationError.errorCode attribute value and the ApplicationError.errorCode attribute of all ApplicationErrors referenced in role possibleError by the ClientServerOperation corresponding to the method determined according to [SWS_CM_10314], the right application error shall be identified.

If this computed ApplicationError.errorCode attribute value does not match any of the ApplicationError.errorCode attributes of all ApplicationErrors referenced in role possibleError by the ClientServerOperation, an UncheckedException shall be raised. ⌋*(RS_SOMEIP_00008)*

[SWS_CM_10359] Deserializing the payload - error response mesages ⌈ Based on the method determined according to [SWS_CM_10314] and the application error determined according to [SWS_CM_10358] the Payload of an error response message shall be deserialized according to section 7.3.1.5. – Therefore the ArgumentDataPrototypes referenced by this ApplicationError in role errorContext shall be deserialized according to their order. ⌋*(RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00008, RS_SOMEIP_00028)*

[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_CM_00345] and [SWS_CM_00346]) or the set_exception (see [SWS_CM_00347]) operation of

[3] The 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 (without errorContext) are raised.

the Promise corresponding to this Future shall be invoked. This will unblock any blocking get, wait, wait_for, and wait_until calls that have been performed on this Future. – The set_value operation shall be invoked in case of a received normal response message using the deserialized payload according to [SWS_CM_10316] as an argument. The set_exception operation shall be invoked in case of a received error response message using the deserialized payload according to [SWS_CM_10359] as an argument. ⌋*(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_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007)*

7.3.1.4 Handling Fields

[SWS_CM_10319] Conditions for sending of a SOME/IP event message ⌈ The sending of a SOME/IP event message shall be initiated 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 cancelled the subscripton 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_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_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00010)*

[SWS_CM_10321] Source of a SOME/IP event message ⌈ The source address and the source port of the SOME/IP event message shall set according to

[SWS_CM_10289]. ⌋(*RS_CM_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_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 (see section 7.3.1.5) 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 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 section 7.3.1.5.

⌋(*RS_CM_00201*, *RS_SOMEIP_00041*, *RS_SOMEIP_00022*, *RS_SOMEIP_00003*, *RS_SOMEIP_00004*, *RS_SOMEIP_00009*)

[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 an `Unchecked Exception` shall be raised. ⌋*(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_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00022)*

[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 section 7.3.1.5. ⌋*(RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00028)*

[SWS_CM_10327] Store the received event data ⌈ The deserialized payload containing the event data shall be stored for retrieval via the `Update`, `GetCachedSamples`, and `Cleanup` methods (see [SWS_CM_00120]) of the respective `Field` class for the event determined according to [SWS_CM_10325]. ⌋*(RS_CM_00202, 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_00203, 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 initiated 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_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.fieldDeploy-`

ment.get.transportProtocol respectively. ⌋(*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_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_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 (see section 7.3.1.5) 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 section 7.3.1.5. The Payload for the request message for the `Get` method will be empty.

⌋(*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*)

[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 `SomeipMethodDeployment`s 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 an `Unchecked Exception` shall be raised. ⌋(*S_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.set`s and `SomeipFieldDeployment.get`s of the `SomeipServiceInterfaceDeployment`, the right method shall be identified. ⌋(*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 section 7.3.1.5. ⌋*(RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028)*

[SWS_CM_10337] Store the received method data ⌈ The received method data shall be stored according to [SWS_CM_10305]. ⌋*(RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[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_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 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_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 initiated upon the return of a registered `SetHandler` resp. `GetHandler` (see [SWS_CM_00114] and [SWS_CM_00116]). ⌋*(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` respetively. ⌋*(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_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_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 (see section 7.3.1.5) 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 section 7.3.1.5.

⌋*(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)*

[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 an Unchecked Exception shall be raised. ⌋(*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_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_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 section 7.3.1.5. ⌋(*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_CM_00345] and [SWS_CM_00346]) of the Promise corresponding to this Future shall be invoked using the deserialized payload as an argument. This will unblock any blocking get, wait, wait_for, and wait_until calls that have been performed on this Future. ⌋(*RS_CM_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_00212*, *RS_CM_00213*, *RS_CM_00215*, *RS_SOMEIP_00007*, *RS_SOMEIP_00009*)

7.3.1.5 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_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_00202*)

This means: Parameters that were not defined in the ServiceInterface used to generate or parameterize 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_00201, RS_CM_00202, RS_CM_00211)*

[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. ⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[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. ⌋*(SRS_Xfrm_00101)*

[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. ⌋*(SRS_Xfrm_00101)*

[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 ImplementationDataType.

[SWS_CM_10037] ⌈ Alignment shall always be calculated from start of SOME/IP message. ⌋*(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 deserialized, the unexpected data shall be discarded. The known fraction shall be considered. ⌋*(RS_CM_00202)*

[SWS_CM_10017] ⌈ If less data than expected shall be deserialized and the data to be deserialized belong to a Field, the initValue should be used if it is defined. Otherwise the data shall be discarded and an Unchecked Exception shall be raised. ⌋*(RS_CM_00202)*

In the following the serialization of different parameters is specified.

7.3.1.5.1 Basic Datatypes

[SWS_CM_10036] ⌈ The SwBaseTypes defined in [9] and according to [TPS_STDT_00067] placed in the package /AUTOSAR_Platform/BaseTypes (e.g., /AUTOSAR_Platform/BaseTypes/uint32) which shall be supported for serialization are listed in Table 7.1. ⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

Type	Description	Size [bit]	Remark
boolean	TRUE/FALSE value	8	FALSE (0), TRUE (1)
uint8	unsigned Integer	8	
uint16	unsigned Integer	16	
uint32	unsigned Integer	32	
uint64	unsigned Integer	64	
sint8	signed Integer	8	
sint16	signed Integer	16	
sint32	signed Integer	32	
sint64	signed Integer	64	
float32	floating point number	32	IEEE 754 binary32 (Single Precision)
float64	floating point number	64	IEEE 754 binary64 (Double Precision)

Table 7.1: SwBaseTypes supported for serialization

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

7.3.1.5.2 Enumeration Datatypes

[SWS_CM_10361] ⌈ Enumeration datatypes shall be serialized according to [SWS_CM_10036] based on their underlying basic datatype (i.e., the Primitive Implementation Data Type according to [SWS_CM_00424]) ⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.3.1.5.3 Structured Datatypes (structs)

[SWS_CM_10042] ⌈ A struct shall be serialized in order of depth-first traversal. ⌋ *(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 and a uint32, 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; therefore, the uint32 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_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_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_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_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_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_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_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_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10253] ⌈ If SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField defines the the data type for the length field of a struct, the data shall be:

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

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

[SWS_CM_00256] ⌈ If TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.sizeOfStructLengthField defines the the data type for the length field of a struct, the data shall be:

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

⌋(*RS_CM_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_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_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.

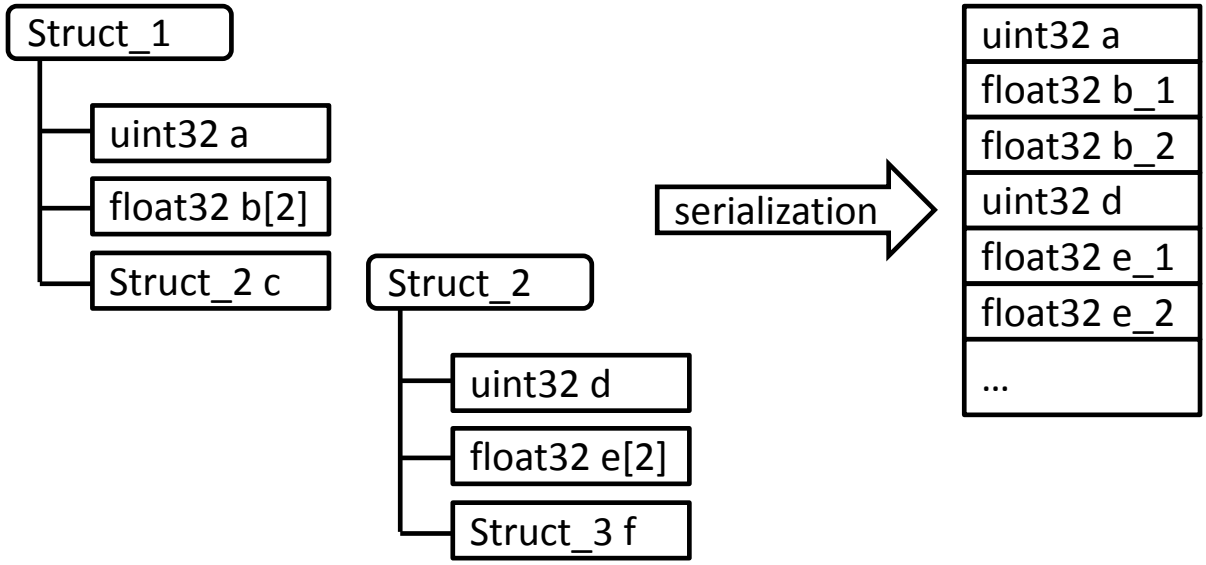

Figure 7.5: Serialization of Structs without Length Fields (Example)

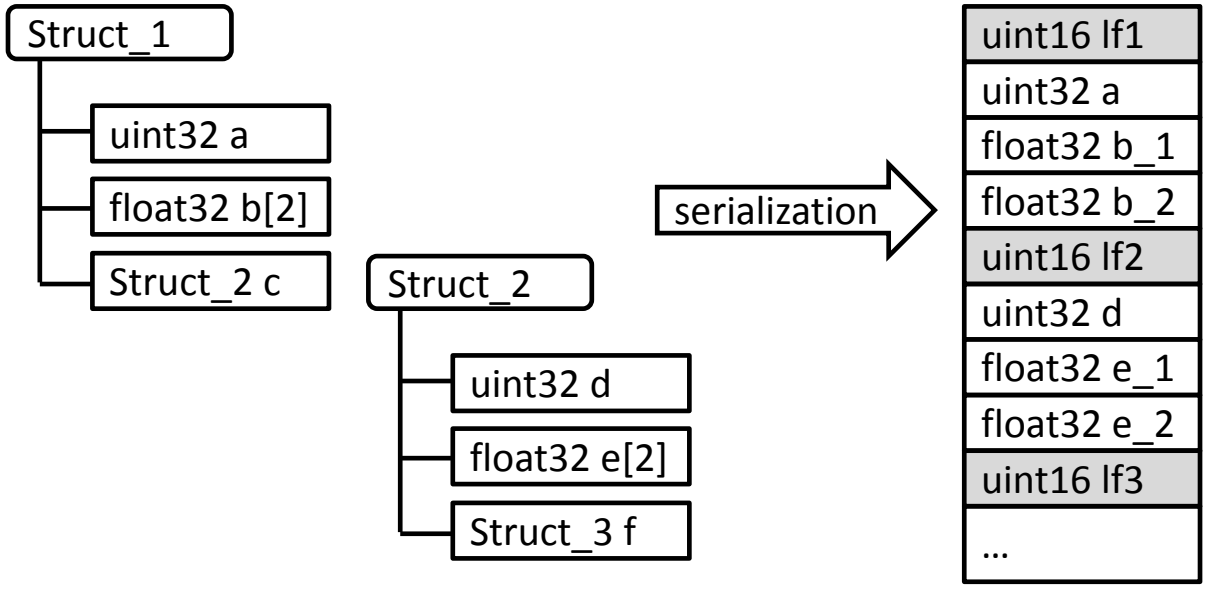

Figure 7.6: Serialization of Structs with Length Fields (Example)

[SWS_CM_01044] ⌈ To define a record element as optional on ServiceInterface level see [TPS_MANI_01083], [TPS_MANI_01084] and [TPS_MANI_01085] for details. ⌋*(RS_CM_00205, RS_SOMEIP_00050)*

[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. ⌈ The serialization of optional record elements shall use the Tag-Length-Value principle, see [3] (chapter 4.1.4.3) for details. ⌋*(RS_CM_00205, RS_SOMEIP_00050)*

[SWS_CM_01046] Regarding the definition of `tlvDataId` see [TPS_MANI_01097] and [constr_1532] for details. ⌈ ⌋*(RS_CM_00205, RS_SOMEIP_00050)*

[SWS_CM_01049] The `tlvDataId`s shall be synchronized between the interacting proxy and skeleton instances. ⌈ The tlvDataId is defined in the context of SomeipDataPrototypeTransformationProps over the attribute SomeipDataPrototypeTransformationProps.dataPrototype.tlvDataId. In other words, the definition is done on the level of specific port instances. Therefore, the tlvDataIds need to be shared between the interacting proxy and skeleton instances, see [TPS_MANI_01097] for details. ⌋*(RS_CM_00205, RS_SOMEIP_00050)*

[SWS_CM_01047] Every record element shall have a `wire type` assigned when the optionality is used for at least one record element inside the struct. ⌈ The wire type determines the wire type (e.g. Base Datatype, Complex Datatype) of the corresponding record element, see [PRS_SOMEIP_00205] for details. ⌋ *(RS_CM_00205, RS_SOMEIP_00050)*

[SWS_CM_01048] Every record element shall have a `tag` assigned when the optionality is used for at least one record element inside the struct. ⌈ The tag shall consist of the wire type and tlvDataId and will be added in front of every optional record element during serialization, see [PRS_SOMEIP_00203] and [TPS_MANI_01097] for details. ⌋*(RS_CM_00205, RS_SOMEIP_00050)*

7.3.1.5.4 Strings

[SWS_CM_10053] ⌈ Strings shall be encoded using Unicode and terminated with a "\0"-character. ⌋*(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_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10285] Responsibility of proper string encoding ⌈ It is the responsibility of the application to provide the string in the encoding defined by AutosarDataType.swDataDefProps.swTextProps.baseType.base-

TypeDefinition.baseTypeEncoding of the respective DataPrototype. The proper encoding will not be checked by the sending SOME/IP network binding implementation during run-time. ⌋(*RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10286] Encoding mismatch in input configurations ⌈ The ara::com generator shall reject input configurations where the SomeipDataPrototypeTransformationProps.networkRepresentation.swTextProps.baseType.baseTypeDefinition.baseTypeEncoding is different from the BaseType.baseTypeDefinition.baseTypeEncoding of the AutosarDataType which is referenced by SomeipDataPrototypeTransformationProps.dataPrototype. ⌋(*RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[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_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10056] ⌈ UTF-16LE and UTF-16BE strings shall have an even length. ⌋ (*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_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_00201*, *RS_CM_00202*, *RS_CM_00211*)

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

For the specification of BOM, see [10] and [11]. Please note that the BOM is used in the serialized strings to achieve compatibility with Unicode.

[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 raising an Unchecked Exception. ⌋(*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_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10242] UTF-8 Strings ⌈ An UTF-8 String shall be represented by an ImplementationDataType

- 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

- or with ImplementationDataType.swDataDefProps.baseType.baseTypeDefinition.baseTypeEncoding set to UTF-8 in case that the DataTypeMap to an ApplicationDataType is missing.

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

[SWS_CM_10243] UTF-16 Strings ⌈ An UTF-16 String shall be represented by an ImplementationDataType

- with category equal to STRING
- which may be mapped to an ApplicationDataType with category equal to STRING using a DataTypeMap
- with AutosarDataType.swDataDefProps.swTextProps.baseType.baseTypeDefinition.baseTypeEncoding set to UTF-16
- or with ImplementationDataType.swDataDefProps.baseType.baseTypeDefinition.baseTypeEncoding set to UTF-16 in case that the DataTypeMap to an ApplicationDataType is missing.

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

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

[SWS_CM_10271] ⌈ 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_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 SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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_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_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_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_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_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_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 `std::string::length()`), including the BOM and any string termination that needs to be added.
2. Appending BOM right after the length field, if BOM is not already available in the first 3 (UTF-8) or 2 (UTF-16) 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. Copying the string data into the output buffer, optionally performing a conversion between UTF-16LE and UTF-16BE between platform and network byte order if `BaseTypeDirectDefinition.byteOrder` and `ApSomeipTransformationProps.byteOrder` have different values
4. 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_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10247] Deserialization of strings ⌈ Deserialization of strings shall consist of the following steps:

1. Check whether the string starts with a BOM. If not, an `Unchecked Exception` shall be raised
2. Check whether BOM has the same value as `ApSomeipTransformationProps.byteOrder`. If not, an `Unchecked Exception` shall be raised
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, an `Unchecked Exception` shall be raised
6. Copy the string data (i.e., everything but the BOM and any string termination added during serialization), optionally performing a conversion between UTF-16LE and UTF-16BE between network and ECU byte order if `BaseTypeDirectDefinition.byteOrder` and `ApSomeipTransformationProps.byteOrder` have different values

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

7.3.1.5.5 Vectors

SOME/IP requires to add a length field of 8, 16 or 32 bit in front of vectors which are treated by SOME/IP as arrays with flexible length. The length field of an array describes the number of bytes of the array.

[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 vector for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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 vector for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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 vector for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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 vector for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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 vector for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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 vector for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10258] ⌈ If attribute TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.sizeOfAr-

rayLengthField 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 vector. ⌋(*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 vector. ⌋(*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 vector, 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_00260] ⌈ If TransformationPropsToServiceInterfaceElementMappingSet.mapping.transformationProps.sizeOfArrayLengthField defines the the data type for the length field of a vector, 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 vector shall be serialized as the concatenation of the following elements:

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

where the size of the length field shall be determined as specified by ApSomeipTransformationProps.sizeOfArrayLengthField which applies to the vector ⌋(*RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10234] ⌈ A vector is represented in adaptive platform by an ImplementationDataType with the category VECTOR and the attribute dynamicArraySizeProfile **not** set. The payload is typed by the ImplementationDataType referenced by subElement ⌋(*RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

In case of nested vectors, the same scheme applies.

[SWS_CM_10222] ⌈ The serializing SOME/IP network binding shall write the size (in bytes) of the serialized vector (without the size of the length field) into the length field. ⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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

7.3.1.5.6 One-dimensional

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

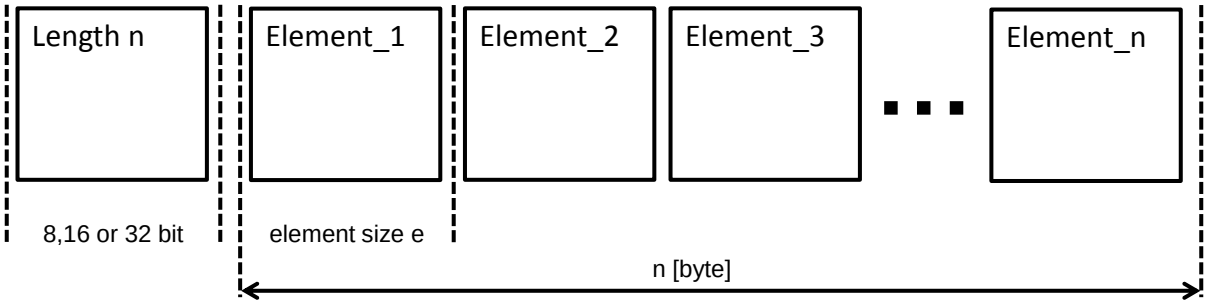

Figure 7.7: One-dimensional vector (Example)

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

7.3.1.5.7 Multi-dimensional

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

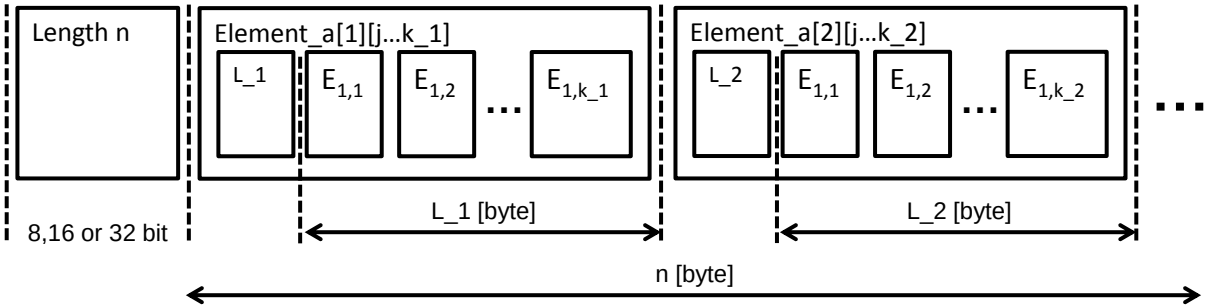

Figure 7.8: Multi-dimensional vector (Example)

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

7.3.1.5.8 Associative Maps

Associative maps are modeled as ApplicationAssocMapDataTypes in the Manifest. As stated in the AUTOSAR Manifest Specification [4] the "natural" language binding for C++ for an associative map is `std::map<key_type,value_type>` where `key_type` is the data type used for the key of the 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 the ApplicationAssocMapElement of the key and the value respectively.

[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_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00262] ⌈ If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField is set to a value equal to 0, no length field shall be inserted in front of the serialized associative map for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋*(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. ⌋*(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_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00263] ⌈ 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 associative map for which the ApSomeipTransformationProps is defined via

SomeipDataPrototypeTransformationProps.someipTransformationProps. ⌋(*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. ⌋(*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_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_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_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_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 *additonal* intermediate padding. Hereby the `key` attribute of an element shall be serialized first followed by the `value` attribute of this element. ⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

Table 7.2 illustrates the serialized form of an exaple 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 = 12 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 after variable length data elements according to [TPS_MANI_03104] 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. ⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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

7.4 Security

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

7.4.1 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] 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_03019)*

[SWS_CM_90001] Restrictions on executing methods ⌈ The invocation of a method by an application shall be executed depending the value of `ClientComSpec.clientCapability`. In case of an invocation without appropriate capability modeling, the network binding of the communication partner which issues the method call shall throw an *UnauthorizedExecutionException* to inform the calling application. ⌋*(RS_SEC_03018, RS_SEC_03019)*

[SWS_CM_90002] Restrictions on sending events ⌈ Sending an event by an application shall be enabled depending on the the value of `SenderComSpec.senderCapability`. In case of triggering an event without appropriate modeling, the network binding of the event sender shall throw an *UnauthorizedEventException* to inform the sending application. ⌋*(RS_SEC_03019)*

[SWS_CM_90003] Restrictions on receiving events ⌈ Receiving an event notification by a subscriber application shall be enabled depending on the the value of `ReceiverComSpec.receiverCapability`. In case of a reception of an event without appropriate capability modeling, the the network binding of the event subscriber shall drop the data. ⌋*(RS_SEC_03019)*

Note:
In case of [SWS_CM_90003] dropping data, the application will not be notified.

A logging facility for security events is currently not defined in the AUTOSAR Adaptive Platform. Logging violations of access restrictions according to [SWS_CM_90001], [SWS_CM_90002] and [SWS_CM_90003] is up to the implementation or specific ECU projects.

7.4.2 Secure Communication

7.4.2.1 SOME/IP

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:

- DTLS

- SecOC

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

It is therefore important to be aware of the limitations of the secure channel approach:

- **Confidentiality of events**
  If events are transported using UDP and may be sent using multicast, they cannot be guaranteed confidential due to the fact that only SecOC can be used to secure multicast communication and SecOC does not offer confidentiality.

[SWS_CM_90101] Secure channel creation ⌈ The Communication Management software shall create secure channels according to the input for all SecureComProps referenced by ServiceInterfaceElementSecureComConfig in the role secureComProps. Secure channels may be shared for multiple communication by multiplexing the communication. ⌋*(RS_SEC_04001)*

[SWS_CM_90102] Using secure 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 role secureComProps of ServiceInterfaceElementSecureComConfig for the AdaptivePlatformServiceInstance referencing the ServiceInterfaceDeployment in the role serviceInterface.

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] TLS secure channel for methods using reliable transport ⌈ A TLS secure channel shall be created and used if:

- **secureComProps**
  A TlsSecureComProps instance is referenced in the role secureComProps by a ServiceInterfaceElementSecureComConfig for the respective method
- **transportProtocol**
  The transportProtocol of the associated SomeipMethodDeployment instance has the value “tcp”

⌋(*RS_SEC_04001*)

[SWS_CM_90104] DTLS secure channel for methods using unreliable transport ⌈ A DTLS secure channel shall be created and used if:

- **secureComProps**
  A TlsSecureComProps instance is referenced in the role secureComProps by a ServiceInterfaceElementSecureComConfig for the respective method
- **transportProtocol**
  The transportProtocol of the associated SomeipMethodDeployment instance has the value “udp”

⌋(*RS_SEC_04001*)

[SWS_CM_90105] TLS secure channel for events using reliable transport ⌈ A TLS secure channel shall be created and used if:

- **secureComProps**
  A TlsSecureComProps instance is referenced in the role secureComProps by a ServiceInterfaceElementSecureComConfig for the respective event
- **transportProtocol**
  The transportProtocol of the associated SomeipEventDeployment instance has the value “tcp”

⌋(*RS_SEC_04001*)

[SWS_CM_90106] DTLS secure channel for events using unreliable transport ⌈ A DTLS secure channel shall be created and used if:

- **secureComProps**
  A TlsSecureComProps instance is referenced in the role secureComProps by a ServiceInterfaceElementSecureComConfig for the respective event
- **transportProtocol**
  The transportProtocol of the associated SomeipEventDeployment instance has the value “udp”
- **multicastThreshold**
  The current number of subscriptions is below the configured multicastThreshold in an SomeipProvidedEventGroup referencing the associated SomeipEventDeployment in the role eventGroup.

⌋(*RS_SEC_04001*)

[SWS_CM_90107] 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)*

A SecOC secure channel may provide authenticity and integrity.

[SWS_CM_90108] SecOC secure channel for methods ⌈ A SecOC secure channel shall be created and used if:

- **secureComProps**
  A `SecOcSecureComProps` instance is referenced in the role `secureComProps` by a `ServiceInterfaceElementSecureComConfig` for the respective `method`

⌋*(RS_SEC_04001)*

[SWS_CM_90109] SecOC secure channel for events ⌈ A SecOC secure channel shall be created and used if:

- **secureComProps**
  A `SecOcSecureComProps` instance is referenced in the role `secureComProps` by a `ServiceInterfaceElementSecureComConfig` for the respective `event`.

⌋*(RS_SEC_04001)*

[SWS_CM_90110] SecOC secure channel for fields ⌈ The requirements [SWS_CM_90108] and [SWS_CM_90109] apply to fields in the same manner, since fields are a composition of `methods` and `events`. ⌋*(RS_SEC_04001)*

8 Communication API specification

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 [12].

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

The header files are not allowed to create objects in memory.

[SWS_CM_01014] No memory allocation in header files ⌈ The header files shall not contain code that creates objects in memory. ⌋*(RS_CM_00001)*

The required folder structure for the ARA public header files is defined by [SWS_AP_00001] in AUTOSAR SWS General [13]. This applies to the *Types header file*, but the folder structure for the *Service header file*s and the *Common header file* is derived from the namespace hierarchy.

[SWS_CM_01020] Folder structure ⌈ The *Service header file*s defined by [SWS_CM_01002] and the *Common header file* defined by [SWS_CM_01012] shall be located within the folder:

`<folder>/<namespace[0]>/<namespace[1]>/.../<namespace[n]>/`

where:
`<folder>` is the start folder for the ara::com header files specific for a project or platform vendor,
`<namespace[0]>` ... `<namespace[n]>` are the namespace names as defined in [SWS_CM_01005]. ⌋*(RS_CM_00001)*

8.1.1.1 Service header files

The *Service header file*s are the central definition of the ara::com API and any associated data structures that are required by the AdaptiveApplication software components to use the communication management.

[SWS_CM_01002] 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)*

[SWS_CM_01004] Inclusion of common header file ⌈ The *Proxy* and *Skeleton header file* shall include the *Common header file*:

```
#include "<name>_common.h"
```

where <name> is the the ServiceInterface.shortName converted to lower-case letters. ⌋*(RS_CM_00001)*

Namespaces are used to separate the definition of services from each other to prevent name conflicts and they allow to use reasonably short names. It is recommended to define the name space unique, e.g. by using the company domain name.

[SWS_CM_01005] Namespace of Service header files ⌈ Based on the symbol attributes of the ordered SymbolProps aggregated by PortInterface in role namespace, the C++ namespace of the *Service header file* shall be:

```
namespace <ServiceInterface.namespace[0].symbol> {
namespace <ServiceInterface.namespace[1].symbol> {
namespace <...> {
namespace <ServiceInterface.namespace[n].symbol> {
...
} // namespace <ServiceInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <ServiceInterface.namespace[1].symbol>
} // namespace <ServiceInterface.namespace[0].symbol>
```

with all namespace names converted to lower-case letters. ⌋*(RS_CM_00002)*

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] Service skeleton namespace ⌈ The C++ namespace for a specific service skeleton class shall be:

```
namespace skeleton {
...
} // namespace skeleton
```

⌋*(RS_CM_00002)*

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

```
namespace proxy {
...
} // namespace proxy
```

⌋*(RS_CM_00002)*

[SWS_CM_01009] Service events namespace ⌈ The *Proxy* and *Skeleton header file* shall provide a C++ namespace for the definition of events within the namespace defined by [SWS_CM_01006] and [SWS_CM_01007] respectively:

```
namespace events {
...
} // namespace events
```

⌋(*RS_CM_00002*)

[SWS_CM_01015] Service methods namespace ⌈ The *Proxy header file* shall provide a C++ namespace for the definition of methods within the namespace defined by [SWS_CM_01007]:

```
namespace methods {
...
} // namespace methods
```

⌋(*RS_CM_00002*)

[SWS_CM_01031] Service fields namespace ⌈ The *Proxy header file* shall provide a C++ namespace for the definition of fields within the namespace defined by [SWS_CM_01006] and [SWS_CM_01007] respectively:

```
namespace fields {
...
} // namespace fields
```

⌋(*RS_CM_00002*)

[SWS_CM_10351] Service application errors ⌈ The *Proxy* and *Skeleton header file* shall provide a C++ namespace for the definition of application errors within the namespace defined by [SWS_CM_01006] and [SWS_CM_01007] respectively:

```
namespace application_errors {
...
} // namespace application_errors
```

⌋(*RS_CM_00002*)

As a summary of the C++ namespace requirements [SWS_CM_01005], [SWS_CM_01006], [SWS_CM_01009], and [SWS_CM_10351], the namespace hierarchy in the *Skeleton header file* looks like:

```
namespace <ServiceInterface.namespace[0].symbol> {
namespace <ServiceInterface.namespace[1].symbol> {
namespace <...> {
namespace <ServiceInterface.namespace[n].symbol> {
namespace skeleton{

namespace events {
...
} // namespace events

namespace application_errors {
...
```

```
} // application_errors

...
} // namespace skeleton
} // namespace <ServiceInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <ServiceInterface.namespace[1].symbol>
} // namespace <ServiceInterface.namespace[0].symbol>
```

As a summary of the C++ namespace requirements [SWS_CM_01005], [SWS_CM_01007], [SWS_CM_01009], [SWS_CM_01015], and [SWS_CM_10351], the namespace hierarchy in the *Proxy header file* looks like:

```
namespace <ServiceInterface.namespace[0].symbol> {
namespace <ServiceInterface.namespace[1].symbol> {
namespace <...> {
namespace <ServiceInterface.namespace[n].symbol> {
namespace proxy{

namespace events {
...
} // namespace events

namespace methods {
...
} // namespace methods

namespace fields {
...
} // namespace fields

namespace application_errors {
...
} // application_errors

...
} // namespace proxy
} // namespace <ServiceInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <ServiceInterface.namespace[1].symbol>
} // namespace <ServiceInterface.namespace[0].symbol>
```

8.1.1.2 Common header file

The *Common header file* includes the ara::com specific type declarations derived from the `ImplementationDataType`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_01012] 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*)

As a minimal requirement, the *Types header file* needs to be included, but there might be additional includes, e.g. for std::string or std::vector, depending on the BaseType of the data type declarations.

[SWS_CM_01001] Inclusion of Types header file ⌈ The *Common header file* shall include the *Types header file*:

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

⌋(*RS_CM_00001*)

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_01016] Data Type definitions for AUTOSAR Data Types in Common header file ⌈ The *Common header file* 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_10370] Data Type definitions for Application Errors in Common header file ⌈ The *Common header file* shall include the class definitions for all sub-classes of ApplicationErrorException for the ApplicationErrors of a ServiceInterface according to [SWS_CM_10356]. ⌋(*RS_CM_00001*)

[SWS_CM_01017] Service 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] Common header file namespace ⌈ The declarations and definitions according [SWS_CM_01016] and [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] Types header file existence ⌈ The communication management shall provide a *Types header file* by using the file name types.h. ⌋(*RS_CM_00001*)

[SWS_CM_01018] Types header file namespace ⌈ The C++ namespace for the data type definitions included by the *Types header file* shall be:

```
namespace ara {
namespace com {
...
} // namespace com
} // namespace ara
```

⌋*(RS_CM_00002)*

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] Data Type declarations in Types header file ⌈ The *Types header file* shall include the data type definitions according to [SWS_CM_00300], [SWS_CM_00301], [SWS_CM_00302], [SWS_CM_00303], [SWS_CM_00304], [SWS_CM_00305], [SWS_CM_00306], [SWS_CM_00307], [SWS_CM_00308], [SWS_CM_00309], [SWS_CM_00310], [SWS_CM_00311], [SWS_CM_00312], [SWS_CM_10352], and [SWS_CM_10354]. ⌋*(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] Service Identifier and Service Version Classes ⌈ The Communication Management shall provide a C++ class named `ServiceInterface.shortName`. The class contains at least a fully qualified name identifier (`ServiceIdentifier`) and a service version (`ServiceVersion`). 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 [14]. These requirements are fulfilled if the operators `operator==`, `operator<`, and `operator=` as well as a `toString()` method is provided.

```
class <ServiceInterface.shortName> {
 public:
  static constexpr ServiceIdentifierType ServiceIdentifier;
  static constexpr ServiceVersionType ServiceVersion;
};

class ServiceIdentifierType {
  bool operator==(const ServiceIdentifierType& other) const;
  bool operator<(const ServiceIdentifierType& other) const;
  ServiceIdentifierType& operator=(const ServiceIdentifierType& other);
  std::string toString() const;
}

class ServiceVersionType {
  bool operator==(const ServiceVersionType& other) const;
  bool operator<(const ServiceVersionType& other) const;
  ServiceVersionType& operator=(const ServiceVersionType& other);
  std::string toString() const;
}
```

⌋(*RS_CM_00200*)

There might exist different instances of exactly the same service in the system. To handle this, an `InstanceIdentifier` is used to identify a specific instance of a service. It is 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],
- on service proxy side, it types the parameter needed to identify the service instance when searching for a specific instance by [SWS_CM_00122] or [SWS_CM_00123].

[SWS_CM_00302] 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, the class method signatures, and the static member `Any` 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 [14] to allow for logging of `InstanceIdentifier`s as well as storing and managing `InstanceIdentifier`s in C++ container classes by the using application. These requirements are fulfilled if the operators `operator==`, `operator<`, and `operator=` as well as a `toString()` method is provided.

```
class InstanceIdentifier {
 public:
  static const InstanceIdentifier Any;

  explicit InstanceIdentifier(std::string value);
  std::string toString() const;
  bool operator==(const InstanceIdentifier& other) const;
  bool operator<(const InstanceIdentifier& other) const;
  InstanceIdentifier& operator=(const InstanceIdentifier& other);
};
```

⌋(*RS_CM_00101*)

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] Find Service Handle ⌈ The Communication Management shall provide the definition of an opaque `FindServiceHandle` with exactly this name. `FindServiceHandle` shall satisfy the `EqualityComparable` requirements accord-

ing 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 [14] 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)*

For example, a definition of FindServiceHandle could look like this:

```
struct FindServiceHandle {
  internal::ServiceId service_id;
  internal::InstanceId instance_id;
  std::uint32_t uid;

  bool operator==(const FindServiceHandle& other) const;
  bool operator<(const FindServiceHandle& other) const;
  FindServiceHandle& operator=(const FindServiceHandle& other);
  ...
};
```

The usage of the API to find service instances, as defined in [SWS_CM_00122] and [SWS_CM_00123], provides a *handle container* holding a list of *handle*s. Each *handle* represents an existing service instance and by passing the *handle* as parameter to the proxy constructor [SWS_CM_00131], it allows the ara::com API user to create a proxy instance to access this service instance.

[SWS_CM_00312] 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, the LessThanComparable requirements according to table 18, and the CopyAssignable requirements according to table 23 of section 17.6.3.1 of [14] to allow storing and managing HandleTypes in C++ container classes by the using application. These requirements are fulfilled if the following operators are provided: operator==, operator<, and operator=.
The definition of the HandleType class shall be located inside the ServiceProxy class defined by [SWS_CM_00004]. This allows the Communication Management software to provide handles with different implementation dependent on the binding to the represented service.

```
class HandleType {
 public:
  bool operator==(const HandleType& other) const;
  bool operator<(const HandleType& other) const;
  HandleType& operator=(const HandleType& other);
  const ara::com::InstanceIdentifier& GetInstanceId() const;
};
```

⌋(*RS_CM_00102*)

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_00304] 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 [14]. A `std::vector` for example fulfills these requirements.

```
template <typename T>
using ServiceHandleContainer = std::vector<T>;
```

⌋(*RS_CM_00102*)

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_00305] 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.

```
template <typename T>
using FindServiceHandler =
std::function<void(ServiceHandleContainer<T>)>;
```

⌋(*RS_CM_00102*)

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 caches. Beside the cache size, the Communication Management requests the update policy of the application local cache for the specific event when subscribing to a specific event by [SWS_CM_00141].

[SWS_CM_00300] Event Cache Update Policy ⌈ The Communication Management shall provide an enumeration `EventCacheUpdatePolicy` which defines the policy of the event cache update. The following policies shall be supported:

- `kLastN`: With this policy, for each call of `Update` the new available events are added to the cache. If they do not fit into the cache, the least recently used entries are discarded first.
- `kNewestN`: With this policy, for each call of `Update` the cache gets cleared first and then filled with the new available events. Even if no event has arrived since the last call to `Update`, the cache gets cleared.

```
enum class EventCacheUpdatePolicy : uint8_t {
  kLastN,
  kNewestN
};
```

⌋(*RS_CM_00202, RS_CM_00203*)

After the receiver subscribed to an event, the method `GetCachedSamples` as defined in [SWS_CM_00171] is used to retrieve the *data samples* of that event. It returns a *Sample Container* containing *Sample Pointer*s to the data samples stored in the event cache. A *Sample Pointer* is an alias for an event data type pointer.

`SamplePtr` behaves as `std::shared_ptr` but it may be implemented differently or with a subset of features. It also contains some an additional method `E2ECheckStatus` of the referred sample.

[SWS_CM_00306] Sample Pointer ⌈ The Communication Management shall provide the definition of `SamplePtr` as a pointer to a data sample. The implementation is allowed to be changed by the Communication Management software provider. ⌋ (*RS_CM_00202, RS_CM_00203*)

[SWS_CM_90432] Functionality of Sample Pointer ⌈ `SamplePtr` shall have behavior of the standard C++ `std::shared_ptr`. ⌋()

[SWS_CM_90420] E2ECheckStatus of a sample ⌈ The `SamplePtr` shall provide the access to the `E2ECheckStatus` of each sample by means of the method `GetE2ECheckStatus`:

```
ara::e2e::state_machine::CheckStatus GetE2ECheckStatus();

```

⌋(*RS_E2E_08534*)

[SWS_CM_00307] Sample Container ⌈ The Communication Management shall provide the definition of `SampleContainer`. The container holds a list of pointers to data samples and is received via event communication. 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 [14]. A `std::vector` for example fulfills these requirements.

```
template <typename T>
using SampleContainer = std::vector<T>;
```

⌋(*RS_CM_00202, RS_CM_00203*)

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_00163]. A *Sample Allocatee Pointer* is an alias for an event data type pointer used both for allocation and data sending.

[SWS_CM_00308] Sample Allocatee Pointer ⌈ The Communication Management shall provide the definition of `SampleAllocateePtr` as a pointer to a data sample allocated by the Communication Management implementation. The implementation is allowed to be changed by the Communication Management software provider.

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

⌋*(RS_CM_00201)*

The event receiver can register an *Event Receive Handler* as a callback to get notified if new event data has arrived. The callback function itself is defined in the event consumer implementation; the *Event Receive Handler* type is just an general purpose function alias for the use in the method `SetReceiveHandler` as defined by [SWS_CM_00181].

[SWS_CM_00309] Event Receive Handler ⌈ The Communication Management shall provide the definition of `EventReceiveHandler` as a function wrapper without parameters for the handler function that gets called by the Communication Management software in case new event data arrives for an event. The event receiver must provide the function implementation which is not required to be re-entrant.
The symbolic name is set; for the alias it is recommended to use the C++ general-purpose polymorphic function wrapper `std::function`, but this is not mandatory and is allowed to be changed by the Communication Management software provider.

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

⌋*(RS_CM_00203)*

The event receiver can monitor the state of a service event subscription by requesting or getting a notification of the *Subscription State*, 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 and a *Subscription State Changed Handler* function wrapper.

[SWS_CM_00310] Subscription State ⌈ The Communication Management shall provide an enumeration `SubscriptionState` which defines the subscription state of an event.

```
enum class SubscriptionState : uint8_t {
  kSubscribed,
  kNotSubscribed,
  kSubscriptionPending
};
```

⌋*(RS_CM_00103, RS_CM_00104)*

[SWS_CM_00311] Subscription State Changed Handler ⌈ The Communication Management shall provide the definition of `SubscriptionStateChangeHandler` as a function wrapper for the handler function that gets called by the Communication Management software in case the subscription state of an event has changed.

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

⌋(*RS_CM_00103*, *RS_CM_00104*)

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] Method Call Processing Mode ⌈ The Communication Management shall provide an enumeration `MethodCallProcessingMode` which defines the processing modes for the service implementation side.

```
enum class MethodCallProcessingMode : uint8_t {
  kPoll,
  kEvent,
  kEventSingleThread
};
```

⌋(*RS_CM_00211*)

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 following section describes the `Future` and `Promise` class templates used in ara::com to provide and retrieve the results of method calls. Whenever there is a mention of a standard C++11 item (class, class template, enum or function) such as `std::future` or `std::promise`, the implied source material is [14]. Whenever there is a mention of an experimental C++ item such as `std::experimental::future::is_ready`, the implied source material is [15].

Futures are technically referred to as "asynchronous return objects", and promises are referred to as "asynchronous providers". Their interaction is made possible by a "shared state". The "shared state" concept is described in [14], section 30.6.4. The description also applies to the shared state behind `ara::com Future` and `Promise`, with the following amendments:

- ", as used by async when policy is launch::deferred" is removed from paragraph 2.

- Paragraph 10, referring to "promise::set_value_at_thread_exit", is removed.

[SWS_CM_00320] **FutureStatus** ⌈ The Communication Management shall provide an enumeration FutureStatus which contains an operation status for timed wait functions of ara::com::Future.

```
enum class FutureStatus : uint8_t {
    ready,
    timeout
};
```

⌋*(RS_CM_00214)*

Note: The meaning of the values is the same as that of the corresponding ones in std::future_status.

[SWS_CM_00321] **Future Class Template** ⌈ The Communication Management shall provide a Future class template which provides a way to check and retrieve results of method calls.

```
template<typename T>
class Future {
  // Default constructor
  Future() noexcept;
  // Move constructor
  Future(Future&&) noexcept;
  // Default copy constructor deleted
  Future(const Future&) = delete;
  // Specialized unwrapping constructor
  Future(Future<Future<T>>&&) noexcept;

  ~Future();

  // Move assignment operator
  Future& operator=(Future&&) noexcept;
  // Default copy assignment operator deleted
  Future& operator=(const Future&) = delete;

  // Returns the result
  T get();

  // Check if the Future has any shared state
  bool valid() const noexcept;

  // Block until the shared state is ready.
  void wait() const;

  // Wait for a specified relative time.
  template< class Rep, class Period >
  FutureStatus wait_for(
    const std::chrono::duration<Rep,Period>& timeout_duration) const;
```

```
  // Wait until a specified absolute time.
  template <class Clock, class Duration>
  FutureStatus wait_until(
    const std::chrono::time_point<Clock,Duration>& abs_time) const;

  // Set a continuation for when the shared state is ready.
  template <typename F>
  auto then(F&& func) -> Future<decltype(func(std::move(*this)))>;

  // Return true only when the shared state is ready.
  bool is_ready() const;
};
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00322] **`Future` default constructor** ⌈ The `Future` constructor

```
Future() noexcept;
```

behaves as the `std::future` constructor

```
future() noexcept;
```

⌋(*RS_CM_00214*)

[SWS_CM_00323] **`Future` move constructor** ⌈ The `Future` constructor

```
Future(Future&&) noexcept;
```

behaves as the `std::future` constructor

```
future(future&&) noexcept;
```

⌋(*RS_CM_00214*)

[SWS_CM_00324] **`Future` unwrapping constructor** ⌈ The `Future` constructor

```
Future(Future<Future<T>>&&) noexcept;
```

behaves as the `std::experimental::future` constructor

```
future(future<future<R>>&&) noexcept;
```

⌋(*RS_CM_00214*)

[SWS_CM_00325] **Move assignment operator** ⌈ The `Future` operator

```
Future& operator=(Future&&) noexcept;
```

behaves as the `std::future` operator

```
future& operator=(future&& rhs) noexcept;
```

⌋(*RS_CM_00214*)

[SWS_CM_00326] **`Future::get`** ⌈ The `Future` function

```
T get();
```

behaves as the `std::future` function

```
R get();
```

⌋(*RS_CM_00214*)

[SWS_CM_00327] **`Future::valid`** ⌈ The `Future` function

```
bool valid() const noexcept;
```

behaves as the `std::future` function

```
bool valid() const noexcept;
```

⌋(*RS_CM_00214*)

[SWS_CM_00328] **`Future::wait`** ⌈ The `Future` function

```
void wait() const;
```

Behaves as the `std::future` function

```
void wait() const;
```

⌋(*RS_CM_00214*)

[SWS_CM_00329] **`Future::wait_for`** ⌈ The `Future` function

```
template< class Rep, class Period >
FutureStatus wait_for(
  const std::chrono::duration<Rep,Period>& timeout_duration) const;
```

behaves as the `std::future` function

```
template <class Rep, class Period>
future_status wait_for(
  const chrono::duration<Rep, Period>& rel_time) const;
```

but using `FutureStatus` instead of `std::future_status`.
Note: The value `std::future_status::deferred` has no correpondent. ⌋
(*RS_CM_00214*)

[SWS_CM_00330] **`Future::wait_until`** ⌈ The `Future` function

```
template <class Clock, class Duration>
FutureStatus wait_until(
  const std::chrono::time_point<Clock,Duration>& abs_time) const;
```

behaves as the `std::future` function

```
template <class Clock, class Duration>
future_status wait_until(
  const chrono::time_point<Clock, Duration>& abs_time) const;
```

but using `FutureStatus` instead of `std::future_status`.
Note: The value `std::future_status::deferred` has no correspondent. ⌋ *(RS_CM_00214)*

[SWS_CM_00331] **`Future::then`** ⌈ The `Future` function

```
template <typename F>
auto then(F&& func) -> Future<decltype(func(std::move(*this)))>;
```

behaves as the `std::experimental::future` function

```
template <class F>
<<see below>> then(F&& func);
```

but without performing *implicit unwrapping*. ⌋*(RS_CM_00215)*

[SWS_CM_00332] **`Future::is_ready`** ⌈ The `Future` function

```
bool is_ready() const;
```

behaves as the `std::experimental::future` function

```
bool is_ready() const;
```

⌋*(RS_CM_00214)*

[SWS_CM_00340] **`Promise` Class Template** ⌈ The Communication Management shall provide a `Promise` class template which provides a way to set a value or exception into the shared state.

```
template <class T>
class Promise {
public:
  // Default constructor
  Promise();
  // Default copy constructor deleted
  Promise(const Promise&) = delete;
  // Move constructor
  Promise(Promise&&) noexcept;

  ~Promise();

  // Default copy assignment operator deleted
  Promise& operator=(const Promise&) = delete;
  // Move assignment operator
  Promise& operator=(Promise&&) noexcept;

  // Return a Future with the same shared state.
  Future<T> get_future();

  // Store an exception in the shared state.
  void set_exception(std::exception_ptr p);
```

```
  // Store a value in the shared state.
  void set_value(const T& value);
  void set_value(T&& value);

  // Set a handler to be called, upon future destruction.
  void set_future_dtor_handler(std::function<void> handler);
};
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00341] `Promise` default constructor ⌈ The `Promise` constructor

```
Promise();
```

behaves as the `std::promise` constructor

```
promise();
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00342] `Promise` move constructor ⌈ The `Promise` constructor

```
Promise(Promise&&) noexcept;
```

behaves as the `std::promise` constructor

```
promise(promise&&) noexcept;
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00343] `Promise` move assignment operator ⌈ The `Promise` operator

```
Promise& operator=(Promise&&) noexcept;
```

behaves as the `std::promise` operator

```
promise& operator=(promise&& rhs) noexcept;
```

Note: the `promise::swap` function the explanation in the standard refers to has no correspondent for `Promise`, but the standard function's behaviour is considered. ⌋ (*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00344] `Promise::get_future` ⌈ The `Promise` function

```
Future<T> get_future();
```

behaves as the `std::promise` function

```
future<R> get_future();
```

but returning a `Future` instead of an `std::future`. ⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00345] `Promise::set_value` ⌈ The `Promise` function

```
void set_value(const T& value);
```

behaves as the `std::promise` function

```
void promise::set_value(const R& r);
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00346] `Promise::set_value`, forwarding reference version ⌈ The `Promise` function

```
void set_value(T&& value);
```

behaves as the `std::promise` function

```
void promise::set_value(R&& r);
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00347] `Promise::set_exception` ⌈ The `Promise` function

```
void set_exception(std::exception_ptr p);
```

behaves as the `std::promise` function

```
void set_exception(exception_ptr p);
```

⌋(*RS_CM_00214*, *RS_CM_00215*)

[SWS_CM_00348] `Promise::set_future_dtor_handler` ⌈ The `Promise` function

```
void set_future_dtor_handler(std::function<void> handler);
```

sets a handler to be called upon destruction of the `Future` associated with the `Promise`'s shared state.
Note: the destruction of the associated `Future` implies the value or exception set by the `Promise` cannot be received from that point on. ⌋(*RS_CM_00214*, *RS_CM_00215*)

8.1.2.4.2 Optional Data Types

The following section describes the `Optional` class template `ara::com::Optional` used in ara::com to provide access to optional record elements of a `Structure Implementation Data Type`. Whenever there is a mention of the standard C++17 Item `std::optional`, the implied source material is [16].

The class template std::optional manages optional record elements, i.e. values that may or may not be present. Every instance of an optional record element either contains a value or does not. The existence can be evaluated during runtime.
Note: Mandatory record elements are declared directly with the corresponding `ImplementationDataType` without using `std::optional`.

[SWS_CM_01033] `Optional` Class Template ⌈ The Communication Management shall at least provide an `Optional` class template which provides a way to check and set the availability of optional record elements.

```
template< class T >
class Optional {

  // Default constructor
  Optional() noexcept;
  // Move constructor
  Optional( Optional&& ) noexcept;
  // Copy constructor
  Optional( const Optional<T>& );

  ~Optional();

  // Move assignment operator
  Optional& operator=(Optional&&) noexcept;
  // Default copy assignment operator
  Optional& operator=(const Optional&);

  // Returns the value
  T& value();

  // Check if the value is available
  bool has_value();

  // Overload bool operator
  operator bool();

  // Destroy value and mark as unavailable
  void reset();

};
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01034] **`Optional` default constructor** ⌈ The `Optional` constructor

```
Optional();
```

behaves as the `std::optional` constructor

```
optional();
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01035] **`Optional` move constructor** ⌈ The `Optional` move constructor

```
Optional( Optional&& ) noexcept;
```

behaves as the `std::optional` move constructor

```
constexpr optional( optional&& other ) noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01036] **`Optional` copy constructor** ⌈ The `Optional` copy constructor

```
Optional( const Optional& );
```

behaves as the `std::optional` copy constructor

```
constexpr optional( const optional& other );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01037] `Optional` **destructor** ⌈ The `Optional` destructor

```
~Optional();
```

behaves as the `std::optional` destructor

```
~optional();
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01038] `Optional` **move assignment operator** ⌈ The `Optional` move assignment operator

```
Optional& operator=(Optional&&) noexcept;
```

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

```
constexpr optional( optional&& other ) noexcept
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01039] `Optional` **default copy assignment operator** ⌈ The `Optional` default copy assignment operator

```
Optional& operator=(const Optional&);
```

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

```
optional& operator=( const optional& other );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01040] `Optional` **function to get contained value** ⌈ The `Optional` function to get contained value

```
T& value();
```

If `*this` contains a value, returns a reference to the contained value.
Otherwise, throws a `std::bad_optional_access` exception. ⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01041] `Optional` **function to check availability of contained value** ⌈ The `Optional` checker function to check the availability of the contained value

```
bool has_value();
```

true if `*this` contains a value, false if `*this` does not contain a value. ⌋(*RS_CM_00205*, *RS_SOMEIP_00050*)

[SWS_CM_01042] `Optional` **bool operator** ⌈ The `Optional` bool operator

```
operator bool();
```

true if `*this` contains a value, false if `*this` does not contain a value. ⌋ *(RS_CM_00205, RS_SOMEIP_00050)*

[SWS_CM_01043] `Optional` reset function ⌈ The `Optional` reset function

```
void reset();
```

If `*this` contains a value, destroy that value as if by `value().T::~T()`. Otherwise, there are no effects.
`*this` does not contain a value after this call. ⌋*(RS_CM_00205, RS_SOMEIP_00050)*

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 exceptions 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 exceptions of the ara::com API are depending on the design of the service. So they are usually generated based on the `DataPrototype`s 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 `ImplementationDataType`, but the Communication Management maps only `ImplementationDataType`s to C++ data types. Therefore it is required in the input configuration that every `ApplicationDataType` used for the typing of a `DataPrototype` is mapped by a `DataTypeMap` to an `ImplementationDataType`.

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

[SWS_CM_00423] Data Type Mapping ⌈ The ara::com generator shall reject input configurations containing a `AutosarDataPrototype` which is typed by an `ApplicationDataType`, but not mapped to an `ImplementationDataType`. ⌋ *(RS_CM_00211)*

The *Common Types Header File* as defined in [SWS_CM_01012] includes the type declarations derived from the `ImplementationDataType`s of the *AUTOSAR Adap-*

tive Platform meta-model classes, depending on the values of the attributes typeEmitter and nativeDeclaration.

[SWS_CM_00421] Provide data type definitions ⌈ The ara::com generator shall provide the corresponding data type definition if the value of attribute typeEmitter is either NOT defined or set to "ARA_COM" and shall silently not generate the data type definition if typeEmitter is set to anything else. ⌋*(RS_CM_00211)*

[SWS_CM_00422] Reject data type definitions ⌈ The ara::com generator shall reject configurations where [SWS_CM_00421] is satisfied, but the ImplementationDataType directly references a SwBaseType without defined nativeDeclaration. ⌋*(RS_CM_00211)*

The redeclaration of C++ types due to the multiple descriptions of equivalent Implementation Data Types in the ServiceInterface description shall be avoided.

[SWS_CM_00411] Avoid Data Type redeclaration ⌈ If there is defined more than one data type with equal Implementation Data Type symbols which are referring to compatible ImplementationDataTypes with identical Implementation Data Type symbols, there shall exist only once the corresponding type declaration as described in the following sub chapters. ⌋*(RS_CM_00211)*

The available meta-model classes are described in detail in the AUTOSAR Manifest Specification [4] and allow to use most of the data types of the *AUTOSAR Classic Platform* like primitive values and structures. Additionally there are *AUTOSAR Adaptive Platform* specific data types available, like string, vector and map.

8.1.2.5.1 Classification of Implementation Data Types

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

- Primitive Implementation Data Type
- Array Implementation Data Type
- Structure Implementation Data Type
- String Implementation Data Type
- Vector Implementation Data Type
- Associative Map Implementation Data Type
- Redefinition Implementation Data Type
- Enumeration Data Type

A Primitive Implementation Data Type is classified either by the category attribute set to VALUE and that it directly refers to a SwBaseType in the role baseType of its SwDataDefProps; or by a Redefinition Implementation Data Type,

which, after all type references have been resolved, boils down to an ImplementationDataType of category VALUE.

An Array Implementation Data Type is classified by the category attribute set to ARRAY and that it defines ImplementationDataTypeElements for each dimension of the array. The arraySize specifies the number of array elements of the dimension.

A Structure Implementation Data Type is classified by the category attribute of the ImplementationDataType set to STRUCTURE and that it has ImplementationDataTypeElements. Each ImplementationDataTypeElement itself can be one of the listed kinds again.

A String Implementation Data Type is classified by the category attribute of the ImplementationDataType set to STRING.
For more details, see chapter 3.3.3.1 of AUTOSAR Manifest Specification [4].

A Vector Implementation Data Type is classified by the category attribute of the ImplementationDataType set to VECTOR and that it has one ImplementationDataTypeElement. The ImplementationDataTypeElement itself can be one of the listed kinds again.
For more details, see chapter 3.3.3.2 of AUTOSAR Manifest Specification [4].

An Associative Map Implementation Data Type is classified by the category attribute of the ImplementationDataType set to ASSOCIATIVE_MAP and that it has two ImplementationDataTypeElements.
For more details, see chapter 3.3.3.3 of AUTOSAR Manifest Specification [4].

A Redefinition Implementation Data Type is classified by the category attribute of the referring ImplementationDataType set to TYPE_REFERENCE and that it refers to an ImplementationDataType in the role implementationDataType of its SwDataDefProps.

An Enumeration Data Type is classified by a Primitive Implementation Data Type or ApplicationPrimitiveDataType 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).

8.1.2.5.2 Naming of Implementation Data Types

The data type name is defined by the Implementation Data Type symbol, which is either the shortName or the value of the symbol attribute of the ImplementationDataType.

[SWS_CM_00400] Naming of data types by short name ⌈ The Implementation Data Type symbol shall be the shortName of the ImplementationDataType if no symbol attribute for this ImplementationDataType is defined. ⌋ *(RS_CM_00211)*

[SWS_CM_00401] Naming of data types by symbol ⌈ The Implementation Data Type symbol shall be the value of the SymbolProps.symbol attribute of the ImplementationDataType if the symbol attribute is defined. ⌋*(RS_CM_00211)*

8.1.2.5.3 Primitive Implementation Data Type

The Communication Management declares C++ types for all Primitive Implementation Data Types defined in the ServiceInterface where the referred BaseType has a nativeDeclaration attribute.

[SWS_CM_00402] Primitive Data Type ⌈ For each Primitive Implementation Data Type with a nativeDeclaration attribute, there shall exist the corresponding type declaration as:

```
using <name> = <nativeDeclaration>;
```

where:

<name> is the Implementation Data Type symbol of the Primitive Implementation Data Type,

<nativeDeclaration> is the nativeDeclaration attribute of the referred BaseType.

⌋*(RS_CM_00211)*

8.1.2.5.4 Array Implementation Data Type

The Communication Management declares C++ types for all Array Implementation Data Types defined in the ServiceInterface. In AUTOSAR Adaptive Platform, the C++ binding of an Array Implementation Data Type could either be implemented as a C-style array or as an std::array. It was chosen to implement it as an std::array, because it avoids several limitations of the C-style arrays, e.g. by having a member size() that provides the size of the 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 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 Implementation Data Type holds elements of Structure 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* Array Implementation Data Type aggregates one ImplementationDataTypeElement which itself is not defined as an Array Implementation Data Type.

[SWS_CM_00403] Array Data Type with one dimension ⌈ For each Array Implementation Data Type with one dimension, there shall exist the corresponding type declaration as:

```
using <name> = std::array<<element>, <size>>;
```

where:

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

<element> is the array element specification. It is defined by the ImplementationDataTypeElement which is aggregated by the Array Implementation Data Type,

<size> is the arraySize of the Array's ImplementationDataTypeElement.

⌋*(RS_CM_00211)*

A *multidimensional* Array Implementation Data Type aggregates one ImplementationDataTypeElement which itself is defined as an Array Implementation Data Type. This means, that the ImplementationDataTypeElement defined as <element> according to [SWS_CM_00403] is again categorized as a Array Implementation Data Type and aggregates one further ImplementationDataTypeElement. This definition describes a *two-dimensional* Array Implementation Data Type; consequently a type with more dimensions is described by just nesting more ImplementationDataTypeElements.

[SWS_CM_00404] Array Data Type with more than one dimension ⌈ For each Array 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 std::array for each additional dimension. The total number of dimensions is equal to the number of nested ImplementationDataTypeElements with category ARRAY plus one for the top level Array Implementation

Data Type. The array element itself is specified by the innermost ImplementationDataTypeElement with category different from ARRAY. ⌋(*RS_CM_00211*)

Please note that [SWS_CM_00404] leads to an std::array type definition where the <size> definitions for each dimension are ordered from the leaf to the root ImplementationDataTypeElement, like e.g.:

```
using My2DimArray = std::array<std::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 ImplementationDataTypeElement, like:

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

8.1.2.5.5 Structure Implementation Data Type

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

[SWS_CM_00405] Structure Data Type ⌈ For each Structure Implementation Data Type, there shall exist the corresponding type declaration as:

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

where:

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

<elements> is the record element specification. For each record element defined by one ImplementationDataTypeElement one record element specification <elements> is defined. The record element specifications are ordered according the order of the related ImplementationDataTypeElements in the input configuration. Sequent record elements are separated with a semicolon.

⌋(*RS_CM_00211*)

[SWS_CM_00413] Element specification typed by Base Type ⌈ Record element specifications <elements> shall exist as

```
<nativeDeclaration> <name>;
```

if the ImplementationDataTypeElement has the category attribute set to VALUE and if it refers to an BaseType. The meaning of the fields is identical to [SWS_CM_00402]. ⌋(*RS_CM_00211*)

[SWS_CM_00414] Element specification typed by Implementation Data Type ⌈ Record element specifications <elements> shall exist as

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

if the ImplementationDataTypeElement has the category attribute set to TYPE_REFERENCE and if it refers to an ImplementationDataType. <type> is the Implementation Data Type symbol of the referred ImplementationDataType and <name> is the shortName of the ImplementationDataTypeElement. ⌋(*RS_CM_00211*)

[SWS_CM_00415] Element specification typed by Array ⌈ Record element specifications <elements> shall exist as

```
std::array<<element>, <size>> <name>;
```

if the ImplementationDataTypeElement has the category attribute set to ARRAY. The meaning of <element>, <size> and <name> is identical to [SWS_CM_00403] and [SWS_CM_00404]. ⌋(*RS_CM_00211*)

[SWS_CM_00416] Element specification typed by Structure ⌈ Record element specifications <elements> shall exist as

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

if the ImplementationDataTypeElement has the category attribute set to STRUCTURE. The meaning and order of the fields is identical to [SWS_CM_00405]. Sequent elements are separated with a semicolon. ⌋(*RS_CM_00211*)

[SWS_CM_00420] Element specification typed by String Data Type with baseTypeSize of 8 ⌈ Record element specifications <elements> shall exist as

```
std::string <name>;
```

if the ImplementationDataTypeElement has the category attribute set to STRING and the baseTypeSize is set to a value of 8.

The meaning of <name> is identical to [SWS_CM_00406]. ⌋(*RS_CM_00211*)

[SWS_CM_00428] Element specification typed by String Data Type with baseTypeSize of 16 ⌈ Record element specifications <elements> shall exist as

```
std::u16string <name>;
```

if the ImplementationDataTypeElement has the category attribute set to STRING and the baseTypeSize is set to a value of 16.

The meaning of <name> is identical to [SWS_CM_00427]. ⌋(*RS_CM_00211*)

[SWS_CM_00418] Element specification typed by Vector ⌈ Record element specifications <elements> shall exist as

```
std::vector<<element>> <name>;
```

if the ImplementationDataTypeElement has the category attribute set to VECTOR. The meaning of <element> and <name> is identical to [SWS_CM_00407] and [SWS_CM_00408]. ⌋(*RS_CM_00211*)

[SWS_CM_00419] Element specification typed by Map ⌈ Record element specifications <elements> shall exist as

```
std::map<<key>, <value>> <name>;
```

if the ImplementationDataTypeElement has the category attribute set to ASSOCIATIVE_MAP. The meaning of <key>, <value> and <name> is identical to [SWS_CM_00409]. ⌋(*RS_CM_00211*)

[SWS_CM_01032] Accessing optional record elements inside a Structure Implementation Data Type that are serialized with the Tag-Length-Value principle. ⌈ For each record element inside a Structure 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::com::Optional<element datatype> <name>;
}
e.g.
struct my_struct{
        ara::com::Optional<bool> my_bool;
}
```

where:

<name> is the shortName of the optional ImplementationDataTypeElement,

<element datatype> is the Implementation Data Type Symbol of the ImplementationDataType of the optional ImplementationDataTypeElement.

ara::com::Optional the template class is specified in paragraph 8.1.2.4.2.

⌋(*RS_CM_00205, RS_SOMEIP_00050*)

8.1.2.5.6 String Implementation Data Type

The Communication Management declares C++ types for all String Implementation Data Types defined in the ServiceInterface. In AUTOSAR Adaptive Platform, the C++ binding of a String Implementation Data Type is implemented by an std::string or by std::u16string.

[SWS_CM_00406] String Data Type with baseTypeSize of 8 ⌈ For each String Implementation Data Type where the baseTypeSize is set to a value of 8, there shall exist the corresponding type declaration as:

```
using <name> = std::string;
```

where <name> is the Implementation Data Type symbol of the String Implementation Data Type. ⌋(*RS_CM_00211*)

[SWS_CM_00427] String Data Type with baseTypeSize of 16 ⌈ For each String Implementation Data Type where the baseTypeSize is set to a value of 16, there shall exist the corresponding type declaration as:

```
using <name> = std::u16string;
```

where <name> is the Implementation Data Type symbol of the String Implementation Data Type. ⌋(*RS_CM_00211*)

8.1.2.5.7 Vector Implementation Data Type

The Communication Management declares C++ types for all Vector Implementation Data Types defined in the ServiceInterface. In AUTOSAR Adaptive Platform, the C++ binding of a Vector Implementation Data Type is always implemented by an std::vector.

A vector definition is based on the following information:

- the data type the vector consists of,
- the number of dimensions.

A Vector 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 *one-dimensional* Vector Implementation Data Type aggregates one ImplementationDataTypeElement which itself is not defined as an Vector Implementation Data Type.

[SWS_CM_00407] Vector Data Type with one dimension ⌈ For each Vector Implementation Data Type having only one dimension, there shall exist the corresponding type declaration as:

```
using <name> = std::vector<<element>>;
```

where:

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

<element> is the vector element specification. It is defined by the ImplementationDataTypeElement which is aggregated by the Vector Implementation Data Type. The ImplementationDataTypeElement itself can be one of the data types allowed for the Adaptive Platform.

⌋(*RS_CM_00211*)

For a *one-dimensional* Vector Implementation Data Type, as it is given as example for the definition of a *Linear Vector Data Type* in [4], the corresponding type declaration would look like this:

```
1 using DynamicDataArrayImplLinear = std::vector<uint16>;
```

A *multidimensional* Vector Implementation Data Type aggregates one ImplementationDataTypeElement which itself is defined as an Vector Imple-

mentation Data Type. This means, that the ImplementationDataTypeElement defined as <element> according to [SWS_CM_00407] is again categorized as a Vector Implementation Data Type and aggregates one further ImplementationDataTypeElement. This definition describes a *two-dimensional* Vector Implementation Data Type; consequently a type with more dimensions is described by just nesting more ImplementationDataTypeElements.

[SWS_CM_00408] Vector Data Type with more than one dimension ⌈ For each Vector Implementation Data Type having more than one dimension, there shall exist the corresponding type declaration according to [SWS_CM_00407] as base where <element> has a nested std::vector for each additional dimension. The total number of dimensions is equal to the number of nested ImplementationDataTypeElements with category VECTOR plus one for the top level Vector Implementation Data Type. The vector element itself is specified by the innermost ImplementationDataTypeElement with category different from VECTOR. ⌋*(RS_CM_00211)*

For a *two-dimensional* Vector Implementation Data Type, as it is given as example for the definition of a *Rectangular Vector Data Type* in [4], the corresponding type declaration would look like this:

```
1 using DynamicDataArrayImplRectangular = std::vector<std::vector<uint16>>;
```

For more details how to model Vector Implementation Data Type, see the chapter *Vector Data Type* of AUTOSAR Manifest Specification document [4].

8.1.2.5.8 Associative Map Implementation Data Type

The Communication Management declares C++ types for all Associative Map Implementation Data Types defined in the ServiceInterface. In AUTOSAR Adaptive Platform, the C++ binding of a Associative Map Implementation Data Type is always implemented by an std::map.

[SWS_CM_00409] Associative Map Data Type ⌈ For each Associative Map Implementation Data Type, there shall exist the corresponding type declaration as:

```
using <name> = std::map<<key>, <value>>;
```

where:

<name> is the Implementation Data Type symbol of the Associative Map Implementation Data Type,

<key> is the map key type specification. It is defined by the first ImplementationDataTypeElement which is aggregated by the Associative Map Implementation Data Type. The ImplementationDataTypeElement itself can be one of the data types allowed for the Adaptive Platform as long as the require-

ments on the key data type imposed by the `std::map` implementation (namely the applicability of `std::less<key>`) are met.

<value> is the mapped value type specification. It is defined by the second `ImplementationDataTypeElement` which is aggregated by the `Associative Map Implementation Data Type`. The `ImplementationDataTypeElement` itself can be one of the data types allowed for the Adaptive Platform.

⌋(*RS_CM_00211*)

For a `Associative Map Implementation Data Type` as it is given as example in chapter *Associative Map Data Type* of [4], the corresponding type declaration would look like this:

```
using MyMap = std::map<uint16, uint8>;
```

For more details how to model `Associative Map Implementation Data Type`, see the chapter *Associative Map Data Type* of AUTOSAR Manifest Specification document [4].

8.1.2.5.9 Redefinition of Implementation Data Type

[SWS_CM_00410] Data Type redefinition ⌈ For each `Redefinition Implementation Data Type` which is typed by an `ImplementationDataType`, there shall exist the corresponding type declaration as:

`using <name> = <type>;`

where:

<name> is the `Implementation Data Type symbol` of the `Redefinition Implementation Data Type`,

<type> is the `Implementation Data Type symbol` of the referred `ImplementationDataType`.

⌋(*RS_CM_00211*)

8.1.2.5.10 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] 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 Implementation Data Type symbol of the Primitive Implementation Data Type,

<type> is the type of the Primitive Implementation Data Type, i.e. the nativeDeclaration attribute of the directly referred BaseType if this nativeDeclaration exists, else the Implementation Data Type symbol of the ImplementationDataType where, after all type references have been resolved, the Primitive Implementation Data Type boils down to.

<enumerator-list> are the enumerators as defined by [SWS_CM_00425].

⌋*(RS_CM_00211)*

The enumerator names base on the CompuScale code symbolic name as defined in [TPS_SWCT_01569] of the AUTOSAR Software Component Template [17].

[SWS_CM_00425] Definition of enumerators ⌈ For each CompuScale 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_00425] is an unsigned data type, or empty if it is a signed data type.

⌋(*RS_CM_00211*)

[SWS_CM_00426] 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::com generator shall reject this input as an invalid configuration. ⌋(*RS_CM_00211*)

8.1.2.6 Error Exception Types

The ara::com API make use of C++ exceptions to notify the user of the API about any errors occurred. ara::com API does hereby strictly follow [18, AUTOSAR CPP14 guidelines] regarding exception usage. I.e. there is a clean seperation of exception types into Checked Exceptions and Unchecked Exceptions, which ara::com API builds upon.

The latter ones (i.e., Unchecked Exceptions) can basically occur in *any* ara::com API call, are not formally modeled in the Manifest, and are fully implementation specific.

The former ones (i.e., Checked Exceptions) can only occur in the context of a call of a service interface method, are formally modeled in the Manifest (as ApplicationErrors), and are fully covered by the AUTOSAR standard.

The types described in this chapter are related to Checked Exceptions and the corresponding ApplicationErrors.

There are two types of Checked Exceptions, which might be thrown in the course of a service method call on the proxy side:

ServiceNotAvailableException: This exception indicates that the Communication Management implementation detected during the processing of a method call, that the providing service instance has already stopped offering the service.

ApplicationErrorException: This exception serves as the base class for all kinds of exceptions related to the ApplicationErrors defined on meta-model level for a specific ServiceInterface. They are created at the application level service provider (skeleton) side and transported to the caller (proxy) side.

[SWS_CM_10352] Definition of ServiceNotAvailableException ⌈ The ServiceNotAvailableException shall be a direct sub-class of std::exception. - Thus the definition should look the following way:

```
1 class ServiceNotAvailableException: public std::exception
2 {
3 };
```

This definition of the ServiceNotAvailableException may be extended by the Communication Management software provider. ⌋(*RS_CM_00102*)

[SWS_CM_10353] Use of `ServiceNotAvailableException` ⌈ The `ServiceNotAvailableException` shall be thrown by the Communication Management implementation in case it detects during processing of a method call, that the providing service instance has already stopped offering the service. ⌋*(RS_CM_00102)*

[SWS_CM_10354] Definition of `ApplicationErrorException` ⌈ The `ApplicationErrorException` shall be a direct sub-class of `std::exception` which shall provide a pure virtual `what` method that shall be overridden by all derived classes.

Thus the definition should look the following way:

```
class ApplicationErrorException: public std::exception
{
public:
        virtual const char *what() const noexcept override = 0;
};
```

This definition of the `ApplicationErrorException` may be extended by the Communication Management software provider. ⌋*(RS_CM_00211)*

[SWS_CM_10355] Use of `ApplicationErrorException` ⌈ The `ApplicationErrorException` shall serve as a base class for all kinds of exceptions related to the `ApplicationError`s defined on meta-model level for a specific `ServiceInterface`. ⌋*(RS_CM_00211)*

[SWS_CM_10356] Definition of sub-classes of `ApplicationErrorException` ⌈ For each `ApplicationError` composed by a `ServiceInterface` in role `possibleError`, a dedicated direct sub-class of `ApplicationErrorException` shall be defined. Hereby the name of this subclass shall be the `shortName` of the `ApplicationError` (`<AE.SN>`).

This sub-class shall override the `what` method in order to return a descriptive error message containing the following information:

- the `shortName` of the `ApplicationError`
- the fully qualified `shortName` of the `ServiceInterface`
- the `errorCode` of the `ApplicationError`

Additionally for every `ArgumentDataPrototype` referenced by the `ApplicationError` in role `errorContext` the sub-class shall contain a dedicated getter method named `get<ADP.SN>` where `<ADP.SN>` is the `shortName` of the `ArgumentDataPrototype`. The return type of this getter method shall be the `Implementation Data Type symbol` (`<IDTS>`) of the `ArgumentDataPrototype` according to [SWS_CM_00401] and [SWS_CM_00400]. - Thus the definition should look the following way:

```
class <ApplicationError.SN>: public ApplicationErrorException
{
public:
  <ApplicationError.SN>(<IDTS0> <ADP0.SN>, <IDTS0> <ADP0.SN> [..]);
  const char *what() const noexcept final override;
  const <IDTS0>& get<ADP0.SN>() const;
```

```
7     const <IDTS1>& get<ADP1.SN>() const;
8     [..]
9   };
```

This definition of the <ApplicationError.SN> may be extended by the Communication Management software provider. ⌋(*RS_CM_00211*)

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] ara::e2e:state_machine::E2E check status ⌈ The Communication Management shall provide an enumeration ara::e2e::state_machine::CheckStatus which represents the results of the check of a single sample:

- Ok: OK: the checks of the sample in this cycle were successful (including counter check).
- Repeated: sample has a repeated counter.
- WrongSequence: The checks of the sample in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
- Error: Error not related to counters occurred (e.g. wrong crc, wrong length, wrong Data ID).
- NotAvailable: No value has been received yet (e.g. during initialization). This is used as the initialization value for the buffer.
- NoNewData: No new data is available (assuming a sample has already been received since the initialization).

```
enum class CheckStatus : uint8_t
{
    Ok,
    Repeated,
    WrongSequence,
    Error,
    NotAvailable,
    NoNewData
};
```

⌋(*RS_E2E_08534*)

The E2E State is determined by checking a history of CheckStatuses.

[SWS_CM_90422] ara::e2e:state_machine::State ⌈ The Communication Management shall provide an enumeration ara::e2e:state_machine::E2EState which represents in what state is the e2e check of the sample(s) of the event. If State is Valid, then the sample(s) can be used.

- `Valid`: Communication of the `sample`s of this `event` functioning properly according to e2e checks, `sample`(s) *can* be used.
- `NoData`: State before `ara::e2e` is initialized, sample cannot be used.
- `Init`: No data from the publisher is available since the initialization, `sample`(s) cannot be used.
- `Invalid`: Communication of the `sample` of this `event` not functioning properly, `sample`(s) *cannot* be used.

```
enum class State : uint8_t
{
    Valid,
    NoData,
    Init,
    Invalid
};
```

⌋(*RS_E2E_08534*)

The E2EResult is a class providing `CheckStatus` and `State`.

[SWS_CM_90423] E2EResult ⌈ The Communication Management shall provide a C++ class named `ara::e2e::e2exf::Result` which provides `ara::e2e:state_machine::State` and `ara::e2e::state_machine::CheckStatus`.

```
class E2EResult {
 public:
 e2e::state_machine::CheckStatus GetCheckStatus() const noexcept;
 e2e::state_machine::State GetState() const noexcept;
};
```

⌋(*RS_E2E_08534*)

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] 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.

```
class <ServiceInterface.shortName>Skeleton {
...
}
```

⌋(*RS_CM_00101*)

[SWS_CM_00003] Service skeleton Event class ⌈ For each VariableDataPrototype defined in the ServiceInterface in the role event the definition of a C++ class using the shortName of the VariableDataPrototype shall be provided in the service skeleton header file within the namespace defined by [SWS_CM_01009].

```
class <VariableDataPrototype.shortName> {
...
}
```

⌋(*RS_CM_00201*)

[SWS_CM_00007] Service skeleton Field class ⌈ For each Field defined in the ServiceInterface in the role field the definition of a C++ class using the shortName of the Field shall be provided in the service skeleton header file within the namespace defined by [SWS_CM_01031].

```
class <Field.shortName> {
...
}
```

⌋(*RS_CM_00219*)

[SWS_CM_00004] 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.

```
class <ServiceInterface.shortName>Proxy {
...
}
```

⌋(*RS_CM_00102*)

[SWS_CM_00005] Service proxy Event class ⌈ For each VariableDataPrototype defined in the ServiceInterface in the role event the definition of a C++

class using the shortName of the VariableDataPrototype shall be provided in the service proxy header file within the namespace defined by [SWS_CM_01009].

```
class <VariableDataPrototype.shortName> {
...
}
```

⌋(*RS_CM_00103*)

[SWS_CM_00006] Service proxy Method class ⌈ For each ClientServerOperation defined in the ServiceInterface in the role method the definition of a C++ class using the shortName of the ClientServerOperation shall be provided in the service proxy header file within the namespace defined by [SWS_CM_01015].

```
class <ClientServerOperation.shortName> {
...
}
```

⌋(*RS_CM_00212*, *RS_CM_00213*)

The following sub-chapters describe the content of the previously defined classes.

8.1.3.1 Offer service

[SWS_CM_00101] Method to offer a service ⌈ The Communication Management shall provide an OfferService method as part of the ServiceSkeleton class to offer a service to applications.

```
void OfferService();
```

⌋(*RS_CM_00101*)

[SWS_CM_00102] Uniqueness of offered service ⌈ The Communication Management shall check the offered service for uniqueness. If the same or another service with the same service ID and instance ID is already registered the Communication Management shall skip further processing. ⌋(*RS_CM_00101*)

[SWS_CM_00103] Protocol where a service is offered ⌈ When a new service is offered by the application the Communication Management shall check over which protocols this service shall be offered. This information is configured in the class of ServiceInterfaceDeployment referencing the offered ServiceInterface in the role serviceInterface. According of the type of the ServiceInterfaceDeployment the Communication Management shall trigger the service offering over respective protocol. ⌋(*RS_CM_00101*)

[SWS_CM_00111] Method to stop offering a service ⌈ The Communication Management shall provide a StopOfferService method as part of the ServiceSkeleton class to stop offering services to applications.

```
void StopOfferService();
```

⌋(*RS_CM_00105*)

8.1.3.2 Service skeleton creation

[SWS_CM_00130] Creation of service skeleton ⌈ 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.
  The identifier shall be unique, so using the same instance identifier for the creation of more than one skeleton instance shall raise an exception.
- `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 instance,
  ara::com::MethodCallProcessingMode mode =
          ara::com::MethodCallProcessingMode::kEvent
);
```

⌋*(RS_CM_00101)*

8.1.3.3 Send event

[SWS_CM_00161] Method to send a service 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. ⌋*(RS_CM_00201)*

[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)*

[SWS_CM_00163] 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 and filled with the respective data. The data is requested by calling the `Allocate` method of the specific `Event` class. By calling the `Send` method it is ensured that the data is 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*)

See [SWS_CM_00308] for the type definition of `SampleAllocateePtr` and ARA-ComAPI explanatory document [1] for more details on the behavior.

8.1.3.4 Provide a service method

[SWS_CM_00191] 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::com::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::com::Future <Method1Output> Method1(
  TypeInputParameter1 input1,
  TypeInputParameter2 input2,
  ...
) = 0;
```

⌋(*RS_CM_00211*)

8.1.3.5 Processing of service methods

[SWS_CM_00198] 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`): 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** (enumeration element `kEvent`): 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** (enumeration element `kEventSingleThread`): 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)*

[SWS_CM_00199] 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::com::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::com::Future` object allows to register a callback function which is invoked when the next pending execution of a method request is finished.

```
ara::com::Future<bool> ProcessNextMethodCall();
```

⌋*(RS_CM_00211)*

[SWS_CM_10362] Raising checked exceptions for application errors ⌈ Whenever on the skeleton side of a service method an `ApplicationError` – according to the interface description in the Manifest – is detected, the sub-class of `Application-`

ErrorException representing this ApplicationError (see [SWS_CM_10356]) simply shall be stored into the ara::com::Promise object, from which the ara::com::Future is returned to the caller. ⌋(*RS_CM_00211*, *RS_CM_00212*, *RS_CM_00213*, *RS_CM_00214*)

8.1.3.6 Registering get handlers for fields

[SWS_CM_00114] 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. This GetHandler (if registered) shall be called by the implementation whenever the Communication Management receives a Get.

```
void RegisterGetHandler(
        std::function<ara::com::Future<FieldType>(
                )> getHandler);
```

⌋(*RS_CM_00218*)

[SWS_CM_00115] Existence of RegisterGetHandler method ⌈ The existence of RegisterGetHandler as part of the Field class shall be controlled by Field.hasGetter. ⌋(*RS_CM_00218*)

8.1.3.7 Registering set handlers for fields

[SWS_CM_00116] 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. This SetHandler (if registered) shall be called by the implementation whenever the Communication Management receives a Set.

```
void RegisterSetHandler(
        std::function<ara::com::Future<FieldType>(
                const FieldType& value)> setHandler);
```

⌋(*RS_CM_00218*)

[SWS_CM_00117] Existence of the RegisterSetHandler method ⌈ The existence of RegisterSetHandler as part of the Field class shall be controlled by Field.hasSetter. ⌋(*RS_CM_00218*)

[SWS_CM_00119] 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*)

[SWS_CM_00128] Ensuring the existence of valid Field values ⌈ If a service containing a `Field` is offered, an `Unchecked Exception` shall be raised, if `Update()` has not been called yet and one or more of the following applies:

- `hasNotifier` = `true`
- `hasGetter` = `true` and a Getter has not yet been registered.

⌋(*RS_CM_00218*)

[SWS_CM_00129] Ensuring existence of SetHandler ⌈ Upon a call to `OfferService()` in a skeleton implementation for a given service, an `Unchecked Exception` shall be raised, if for at least one contained `Field` having `hasSetter` = `true` no SetHandler has been registered yet. ⌋(*RS_CM_00218*)

8.1.3.8 Find service

[SWS_CM_00121] Method to find a service ⌈ The Communication Management shall provide a `FindService` method as part of the `ServiceProxy` class to enable applications to find services. To support event-based and time-triggered systems the `FindService` method shall be provided in a handler registration and a immediately returned request style. ⌋(*RS_CM_00102*)

[SWS_CM_00122] Find service with immediately returned request ⌈ The `FindService` method of the `ServiceProxy` class with immediately returned request takes an instance ID qualifying the wanted `instance` of the service as optional input parameter. If no `instance` is specified, any instance of the service matches.
As result a container containing handles for all matching service instances is returned.

```
static ara::com::ServiceHandleContainer<<ProxyClassName>::HandleType>
  FindService(ara::com::InstanceIdentifier instance =
    ara::com::InstanceIdentifier::Any);
```

where `<ProxyClassName>` is the name of the `ServiceProxy` class as defined in [SWS_CM_00004]. ⌋(*RS_CM_00102*)

For the definition of the types used in the `StartFindService` signature, see:

- [SWS_CM_00304] for `ServiceHandleContainer`,
- [SWS_CM_00312] for `HandleType`,
- [SWS_CM_00302] for `InstanceIdentifier`.

[SWS_CM_00123] Find service with handler registration ⌈ 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 optionally an instance ID qualifying the wanted `instance` of the service. If no `instance` is spec-

ified any instance of the service matches. 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::com::InstanceIdentifier instance =
    ara::com::InstanceIdentifier::Any);
```

where <ProxyClassName> is the name of the ServiceProxy class as defined in [SWS_CM_00004]. ⌋(*RS_CM_00102*)

For the definition of the types used in the StartFindService signature, see:

- [SWS_CM_00303] for FindServiceHandle,
- [SWS_CM_00305] for FindServiceHandler,
- [SWS_CM_00312] for HandleType,
- [SWS_CM_00302] for InstanceIdentifier.

[SWS_CM_00124] Find service handler behavior ⌈ 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 a new matching service instance is found. ⌋(*RS_CM_00102*)

[SWS_CM_00125] 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*)

See [SWS_CM_00303] for the type definition of FindServiceHandle.

8.1.3.9 Service proxy creation

[SWS_CM_00131] 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(HandleType &handle);
```

⌋(*RS_CM_00102*)

See [SWS_CM_00312] for the type definition of HandleType.

8.1.3.10 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 parameters the `policy` regarding cache update and the `cacheSize` of the subscription needs to be specified. Possible event cache update policies are `ara::com::EventCacheUpdatePolicy::kLastN` and `ara::com::EventCacheUpdatePolicy::kNewestN`. With the last policy the cache always contains the last n received events. Where n is equal to the `cacheSize`. The cache will contain less events until n events have been received.

```
void Event::Subscribe(
  ara::com::EventCacheUpdatePolicy policy,
  size_t cacheSize
);
```

⌋*(RS_CM_00103)*

See [SWS_CM_00300] for the type definition of `EventCacheUpdatePolicy` and ARAComAPI explanatory document [1] for more details on the behavior.

[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)*

8.1.3.11 Receive event using polling

[SWS_CM_00171] Receive a service event using polling ⌈ Inside the specific `Event` class belonging to the specific `ServiceProxy` class, an `Update`, a `GetCachedSamples` and a `Cleanup` method shall be provided to allow for polling of received events.

By calling the `Update` method the event cache is updated with the meanwhile received events. As input parameter the `Update` method allows to specify a `FilterFunction` to throw away received events.

```
bool Event::Update(
        ara::com::FilterFunction<SampleType> filter = {});
```

After updating the event cache via the `Update` method, the current data in the event cache can be retrieved by calling the `GetCachedSamples` method. The return value will be a container containing the events stored in the event cache.

```
const ara::com::SampleContainer<ara::com::SamplePtr<const SampleType>>
 &GetCachedSamples() const;
```

For the definition of the types used in the `GetCachedSamples` signature, see:

- [SWS_CM_00307] for `SampleContainer`,
- [SWS_CM_00306] for `SamplePtr`.

Finally the event cache can be cleaned-up after processing by calling the `Cleanup` method. The `Cleanup` method removes all events from the event cache if the selected caching policy is `ara::com::EventCacheUpdatePolicy::kNewestN`. Otherwise calling the `Cleanup` method has no effect.

```
void Event::Cleanup()
```

⌋*(RS_CM_00202)*

For the e2e-protected `event`s, after updating the event cache via the `Update` method, and before calling `GetCachedSamples`, the current `E2EResult` needs to be retrieved by calling the `GetE2EResult` method.

[SWS_CM_90424] Provide E2E Result ⌈ Inside the specific e2e-protected `Event`s belonging to the specific `ServiceProxy` class, the method `GetE2EResult` shall be provided.

```
const ara::e2e::e2exf::Result GetE2EResult() const;
```

For the definition of the type returned by `GetE2EResult` signature, see:

- [SWS_CM_90423] for `E2EResult`

⌋*(RS_E2E_08534)*

[SWS_CM_00266] FilterFunction for incoming event filtering ⌈ The `FilterFunction` takes as input the received event and decides whether to store or throw away the event. By returning `true` the event is stored for further processing.

```
template<typename S>
using FilterFunction = std::function<bool(const S& sample)>
```

⌋*(RS_CM_00202)*

8.1.3.12 Receive event by getting triggered

[SWS_CM_00181] 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 `Update`, `Get`, and `Cleanup` methods of the specific `Event` class to access the retrieved event data. See [SWS_CM_00309] for its definition. ⌋*(RS_CM_00203)*

[SWS_CM_00182] 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_00183] 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 a `UnsetReceiveHandler` method shall be provided to allow for disabling of triggering the application.

```
void Event::UnsetReceiveHandler()
```

⌋*(RS_CM_00203)*

8.1.3.13 Call a service method

[SWS_CM_00196] Initiate a method call ⌈ The `operator()` shall be provided inside the specific `Method` class belonging to the specific `ServiceProxy` class to allow the call of a method provided by a server.
As input parameters, the `operator()` shall take the respective input parameters of the provided method.
An `Output` type combining the possible output parameters and optional return values shall be provided inside the specific `Method` class belonging to the specific `ServiceProxy` class.
The `operator()` shall return an `ara::com::Future` object wrapping the output parameters and return values.
At the point of time when the caller calls the method, 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::com::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.

```
struct Method1::Output {
  TypeOutputParameter1 output1;
  TypeOutputParameter2 output2;
  ...
  TypeResult result;
}

ara::com::Future<Method1::Output> Method1::operator()(
  TypeInputParameter1 input1,
  TypeInputParameter2 input2,
  ...
);
```

⌋*(RS_CM_00212, RS_CM_00213)*

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::com::Future` object which is used to query for method completion and result including a possibly thrown exception.

[SWS_CM_00194] Cancel the method call ⌈ The `destructor` of the returned `ara::com::Future` object shall be used by the caller to cancel the request after issuing a method call. Deleting the returned `ara::com::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)*

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] Retrieving results of the method call ⌈ The method `get()` of the returned `ara::com::Future` object shall be used to retrieve the result of the method call or to obtain any exception thrown by the method. The call of method `get()` 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` type or throwing an exception. ⌋*(RS_CM_00212)*

[SWS_CM_00192] Synchronous behavior of method call ⌈ To achieve synchronous behavior of the method call, the methods of `ara::com::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)*

[SWS_CM_00193] 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::com::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)*

Note:
When the user just calls `is_ready()` of `ara::com::Future` and on positive response, finally `get()` of `ara::com::Future`, retrieving the result of the method call or any exception thown by the method works polling-based without any overhead in the middleware and uncontrolled context switches due to asynchronous event-style mechanisms.

[SWS_CM_00197] Asynchronous behavior of method call with notification ⌈ To achieve asynchronous behavior of the method call with event-driven notification on the result availability, the non-blocking method `then()` of `ara::com::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)*

[SWS_CM_10371] Context of thrown checked exceptions ⌈ If during processing of a method call one of the checked exceptions (see section subsubsection 8.1.2.6)

occurs, the corresponding checked exception (i.e., either `ServiceNotAvailableException` or the proper sub-class of `ApplicationErrorException`) shall be thrown in the context of the `ara::com::Future::Get()` call. ⌋(*RS_CM_00211*, *RS_CM_00212*, *RS_CM_00213*, *RS_CM_00214*)

8.1.3.14 Get method for fields

[SWS_CM_00112] 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::com::Future<FieldType> Get();
```

⌋(*RS_CM_00218*)

[SWS_CM_00132] 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*)

8.1.3.15 Set method for fields

[SWS_CM_00113] 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::com::Future<FieldType> Set(const FieldType& value);
```

⌋(*RS_CM_00217*)

[SWS_CM_00133] 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*)

8.1.3.16 Update notification events for fields

[SWS_CM_00120] 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
- [SWS_CM_00151] Method to unsubscribe from a service event
- [SWS_CM_00171] Receive a service event using polling
- [SWS_CM_00181] Enable service event trigger
- [SWS_CM_00182] Event Receive Handler call serialization
- [SWS_CM_00183] Disable service event trigger

Except that the corresponding methods reside in the `Field` class instead of the `Event` class. ⌋*(RS_CM_00218)*

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 but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**AdaptivePlatformServiceInstance (abstract)**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	This meta-class represents the ability to describe the existence and configuration of a service instance in an abstract way. **Tags:** atp.Status=draft			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
e2eEventProtectionProps	End2EndEventProtectionProps	*	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 serviceInterface. **Tags:** atp.Status=draft
secureComConfig	ServiceInterfaceElementSecureComConfig	*	aggr	Configuration settings to secure the communication of ServiceInterface elements. **Tags:** atp.Status=draft
serviceInterface	ServiceInterfaceDeployment	0..1	ref	Reference to a ServiceInterfaceDeployment that identifies the ServiceInterface that is represented by the ServiceInstance. **Tags:** atp.Status=draft

Table A.1: AdaptivePlatformServiceInstance

Class	**ApSomeipTransformationProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::TransformationConfiguration			
Note	SOME/IP serialization properties. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, TransformationProps			
Attribute	***Type***	***Mul.***	***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.
sessionHandling	SOMEIPTransformerSessionHandlingEnum	0..1	attr	Defines whether the SOME/IP transformer shall use session handling for Sender/Receiver communication.

Attribute	Type	Mul.	Kind	Note
sizeOfArrayLengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of an Array. It describes the size of the length field (in Bytes) that will be put in front of the Array in the SOME/IP message. In contrast to Classic AUTOSAR this attribute defines the value for both, fixed-size and dynamic-size arrays.
sizeOfStringLengthField	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.
sizeOfStructLengthField	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.
sizeOfUnionLengthField	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.
sizeOfUnionTypeSelectorField	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.

Table A.2: ApSomeipTransformationProps

Class	ApplicationArrayDataType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which is an array, each element is of the same application data type. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement, ARObject, ApplicationCompositeDataType, ApplicationDataType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArrayElement	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.3: ApplicationArrayDataType

Class	**ApplicationAssocMapDataType**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::DataTypes			
Note	An application data type which is a map and consists of a key and a value **Tags:** atp.Status=draft; atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement, ARObject, ApplicationCompositeDataType, ApplicationDataType, Atp Blueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, Collectable Element, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
key	ApplicationAssocMapElement	1	aggr	Key element of the map that is used to uniquely identify the value of the map. **Tags:** atp.Status=draft
value	ApplicationAssocMapElement	1	aggr	Value element of the map that stores the content associated to a key. **Tags:** atp.Status=draft

Table A.4: ApplicationAssocMapDataType

Class	**ApplicationAssocMapElement**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::DataTypes			
Note	Describes the properties of the elements of an application map data type. **Tags:** atp.Status=draft			
Base	ARObject, ApplicationCompositeElementDataPrototype, AtpFeature, AtpPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.5: ApplicationAssocMapElement

Class	**ApplicationDataType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc. It should be possible to model the application level aspects of a VFB system by using ApplicationDataTypes only.			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.6: ApplicationDataType

Class	**ApplicationError**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***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).
errorContext	ArgumentDataPrototype	*	ref	This reference identifies out arguments that shall have a meaning only if an error occurs. **Tags:** atp.Status=draft; atp.StatusComment=Reserved for AUTOSAR adaptive platform

Table A.7: ApplicationError

Class	**ApplicationPrimitiveDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	A primitive data type defines a set of allowed values. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement, ARObject, ApplicationDataType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table A.8: 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***	***Mul.***	***Kind***	***Note***

Attribute	Type	Mul.	Kind	Note
element (ordered)	ApplicationRecordElement	1..*	aggr	Specifies an element of a record. The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordDataType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.9: 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**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.10: 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	ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
direction	ArgumentDirectionEnum	1	attr	This attribute specifies the direction of the argument prototype.
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures and to the value useArrayBaseType for arrays.

Table A.11: ArgumentDataPrototype

Enumeration	ArgumentDirectionEnum
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

Note	Use cases: • Arguments in ClientServerOperation can have different directions that need to be formally indicated because they have an impact on how the function signature looks like eventually. • Arguments in BswModuleEntry already determine a function signature, but the direction is used to specify the semantics, especially of pointer arguments.
Literal	***Description***
in	The argument value is passed to the callee. **Tags:** atp.EnumerationValue=0
inout	The argument value is passed to the callee but also passed back from the callee to the caller. **Tags:** atp.EnumerationValue=1
out	The argument value is passed from the callee to the caller. **Tags:** atp.EnumerationValue=2

Table A.12: ArgumentDirectionEnum

Class	**AutosarDataPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	ARObject, AtpFeature, AtpPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
type	AutosarDataType	1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table A.13: AutosarDataPrototype

Class	**AutosarDataType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for user defined AUTOSAR data types for ECU software.			
Base	ARElement, ARObject, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this AutosarDataType.

Table A.14: 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	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
baseType Definition	BaseTypeDefinition	1	aggr	This is the actual definition of the base type. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table A.15: BaseType

Class	**BaseTypeDirectDefinition**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	ARObject, BaseTypeDefinition			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
baseType Encoding	BaseTypeEncodingString	1	attr	This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **Tags:** xml.sequenceOffset=90
baseType Size	PositiveInteger	0..1	attr	Describes the length of the data type specified in the container in bits. **Tags:** xml.sequenceOffset=70
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:** xml.sequenceOffset=110
maxBaseTypeSize	PositiveInteger	0..1	attr	Describes the maximum length of the BaseType in bits. **Tags:** xml.sequenceOffset=80
memAlignment	PositiveInteger	0..1	attr	This attribute describes the alignment of the memory object in bits. E.g. "8" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. If the value is set to "0" the meaning shall be interpreted as "unspecified". **Tags:** xml.sequenceOffset=100

Attribute	*Type*	*Mul.*	*Kind*	*Note*

nativeDeclaration	NativeDeclarationString	0..1	attr	This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example
				
				BaseType with
				`shortName: "MyUnsignedInt"`
				`nativeDeclaration: "unsigned short"`
				
				Results in
				`typedef unsigned short MyUnsignedInt;`
				
				If the attribute is not defined the referring ImplementationDataTypes will not be generated as a typedef by RTE.
				
				If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseTypeSize.
				
				This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems.
				
				Tags: xml.sequenceOffset=120

Table A.16: 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***
mostSignificantByteFirst	Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format) **Tags:** atp.EnumerationValue=0
mostSignificantByteLast	Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format) **Tags:** atp.EnumerationValue=1

opaque	For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details. **Tags:** atp.EnumerationValue=2

Table A.17: ByteOrderEnum

Class	**ClientComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Client-specific communication attributes (RPortPrototype typed by ClientServerInterface).			
Base	ARObject, RPortComSpec			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
clientCapability	ClientCapabilityEnum	0..1	attr	This attribute represents the expressed capability of the client. The client may decide to claim that existing resources of a ServiceInterface are expressly not used by this specific client. The conceptual background of this claim may be driven by security, safety, etc. **Tags:** atp.Status=draft
getter	Field	0..1	ref	The existence of this reference indicates that the ClientComSpec refers to the getter of a Field. **Tags:** atp.Status=draft
operation	ClientServerOperation	0..1	ref	This represents the corresponding ClientServerOperation.
setter	Field	0..1	ref	The existence of this reference indicates that the ClientComSpec refers to the setter of a Field. **Tags:** atp.Status=draft
transformationComSpecProps	TransformationComSpecProps	*	aggr	This references the TransformationComSpecProps which define port-specific configuration for data transformation.

Table A.18: ClientComSpec

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	ARObject, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime

Attribute	Type	Mul.	Kind	Note
fireAndForget	Boolean	0..1	attr	This attribute defines whether this method is a fire&forget method (true) or not (false). **Tags:** atp.Status=draft
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation.

Table A.19: ClientServerOperation

Class	CompositionDataPrototypeRef			
Package	M2::AUTOSARTemplates::AdaptivePlatform::General			
Note	This meta-class represents the ability to refer to an AUTOSAR DataPrototype in the context of a CompositionSwComponentType. **Tags:** atp.Status=draft			
Base	ARObject			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataPrototype	DataPrototype	0..1	iref	This attribute shall exist if the InstanceRef points to a DataPrototype typed by an ApplicationDataType. **Tags:** atp.Status=draft
elementInImplDatatype	ElementInImplementationDatatypeInstanceRef	0..1	aggr	This attribute shall exist if the InstanceRef points to a DataPrototype typed by an ImplementationDataType. **Tags:** atp.Status=draft
tlvDataId	PositiveInteger	0..1	attr	This attribute represents the ability to specify a TLV data-id for the serialization of a specific DataPrototype in the context of a (potentially deeply-nested) composite data structure for the case that the data structure has optional elements. This value does not represent the entire value of the tag, e.g. the wire-type is not included (because it can be derived from the information about the underlying AutosarDataType).

Table A.20: CompositionDataPrototypeRef

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**	**Mul.**	**Kind**	**Note**

Attribute	Type	Mul.	Kind	Note
compuConstContentType	CompuConstContent	1	aggr	This is the actual content of the constant compu method scale. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=10; xml.typeElement=false; xml.typeWrapperElement=false

Table A.21: CompuConst

Class	CompuConstTextContent			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the textual content of a scale.			
Base	ARObject, CompuConstContent			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
vt	VerbatimString	1	attr	This represents a textual constant in the computation method.

Table A.22: CompuConstTextContent

Class	CompuMethod			
Package	M2::MSR::AsamHdo::ComputationMethod			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. 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. **Tags:** atp.recommendedPackage=CompuMethods			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
compuInternalToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:** xml.sequenceOffset=80
compuPhysToInternal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:** xml.sequenceOffset=90
displayFormat	DisplayFormatString	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:** xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:** xml.sequenceOffset=30

Table A.23: CompuMethod

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***	***Mul.***	***Kind***	***Note***
desc	MultiLanguage OverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:** xml.sequenceOffset=30
compuInverseValue	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
compuScaleContents	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
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
symbol	CIdentifier	0..1	attr	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. **Tags:** xml.sequenceOffset=25

Attribute	Type	Mul.	Kind	Note
upperLimit	Limit	0..1	attr	This specifies the upper limit of a of the scale. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=50

Table A.24: 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**	**Mul.**	**Kind**	**Note**
compuScale (ordered)	CompuScale	*	aggr	This represents one scale within the compu method. Note that it contains a Variationpoint in order to support blueprints of enumerations. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false

Table A.25: CompuScales

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			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
swDataDefProps	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table A.26: DataPrototype

Class	DataTypeMap			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing ImplementationDataType.			
Base	ARObject			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
applicationDataType	ApplicationDataType	1	ref	This is the corresponding ApplicationDataType
implementationDataType	ImplementationDataType	1	ref	This is the corresponding ImplementationDataType.

Attribute	Type	Mul.	Kind	Note

Table A.27: 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**	**Mul.**	**Kind**	**Note**
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an ApplicationDataType and its ImplementationDataType.
modeRequestTypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an ModeDeclarationGroup and its ImplementationDataType.

Table A.28: DataTypeMappingSet

Class	E2EProfileConfiguration			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::E2E			
Note	This element holds E2E profile specific configuration settings. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
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.
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.
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 MaxDeltaCounter 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 WindowSize 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 WindowSize checks, for the state E2E_SM_INVALID.

Attribute	Type	Mul.	Kind	Note
maxErrorStateValid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize 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.
windowSize	PositiveInteger	0..1	attr	Size of the monitoring window for the E2E state machine.

Table A.29: E2EProfileConfiguration

Class	End2EndEventProtectionProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::E2E			
Note	This element allows to protect an event or a field notifier with an E2E profile. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataId (ordered)	PositiveInteger	*	attr	This represents a unique numerical identifier for the referenced event or field notifier that is included in the CRC calculation. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEndProtection.
e2eProfileConfiguration	E2EProfileConfiguration	0..1	ref	Reference to E2E profile configuration settings that are valid to protect the referenced event or field notifier. **Tags:** atp.Status=draft
event	ServiceEventDeployment	0..1	ref	Reference to an event that is protected by the E2E profile. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
maxDataLength	PositiveInteger	0..1	attr	Maximum length of Data in bits.
minDataLength	PositiveInteger	0..1	attr	Minimum length of Data in bits.
notifier	ServiceFieldDeployment	0..1	ref	Reference to a field notifier that is protected by an E2E profile. **Tags:** atp.Status=draft

Table A.30: End2EndEventProtectionProps

Class	EndToEndTransformationComSpecProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	The class EndToEndTransformationIComSpecProps specifies port specific configuration properties for EndToEnd transformer attributes.			
Base	ARObject, Describable, TransformationComSpecProps			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
disableEndToEndCheck	Boolean	1	attr	Disables/Enables the E2E check. The E2Eheader is removed from the payload independent from the setting of this attribute.
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 MaxDeltaCounter 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 WindowSize checks, for the state E2E_SM_INIT. The minimum value is 0.
maxErrorStateInvalid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_INVALID. The minimum value is 0.
maxErrorStateValid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize 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.

Attribute	Type	Mul.	Kind	Note
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.
windowSize	PositiveInteger	0..1	attr	Size of the monitoring window for the E2E state machine. The meaning is the number of correct cycles (E2E_P_OK) that are required in E2E_SM_INITCOM before the transition to E2E_SM_VALID. The minimum allowed value is 1.

Table A.31: EndToEndTransformationComSpecProps

Class	EthernetCommunicationConnector			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet specific attributes to the CommunicationConnector.			
Base	ARObject, CommunicationConnector, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**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.

Attribute	Type	Mul.	Kind	Note
pncFilterDataMask	PositiveUnlimitedInteger	0..1	attr	Bit mask for Ethernet Payload used to configure the Ethernet Transceiver for partial network wakeup. This attribute should not be computed from the pncIdentifier values in order to support future introduction of additional PNCs. Note that for one EcuInstance all contributing pncFilterDataMask will be bitwise ORed to obtain the value of UdpNmPnFilterMaskByte. Note that this data mask is calculated over the whole payload (8 Byte) of the NmPdu ignoring the leading bytes which do not contain pncVector information. The number of leading bytes which shall be ignored is equivalent to the value of System.pncVectorOffset.
unicastNetworkEndpoint	NetworkEndpoint	0..1	ref	Network Endpoint that defines the IPAddress of the machine. **Tags:** atp.Status=draft

Table A.32: EthernetCommunicationConnector

Class	Field			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to define a piece of data that can be accessed with read and/or write semantics. It is also possible to generate a notification if the value of the data changes. **Tags:** atp.Status=draft			
Base	ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**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.
initValue	ValueSpecification	1	aggr	Specifies initial value(s) of the Field. **Tags:** atp.Status=draft

Table A.33: Field

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			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
desc	MultiLanguage OverviewParagraph	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
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
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
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

Attribute	Type	Mul.	Kind	Note
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.34: 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	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**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.
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Attribute	Type	Mul.	Kind	Note
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
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.35: ImplementationDataType

Class	ImplementationDataTypeElement			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • 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	ARObject, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this ImplementationDataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
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.

Attribute	Type	Mul.	Kind	Note
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElementt.

Table A.36: 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			
Attribute	**Type**	**Mul.**	**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.37: ImplementationProps

Class	Ipv4Configuration			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Internet Protocol version 4 (IPv4) configuration.			
Base	ARObject, NetworkEndpointAddress			
Attribute	**Type**	**Mul.**	**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.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
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.38: Ipv4Configuration

Class	**Ipv6Configuration**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Internet Protocol version 6 (IPv6) configuration.			
Base	ARObject, NetworkEndpointAddress			
Attribute	***Type***	***Mul.***	***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.39: Ipv6Configuration

Class	**Machine**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::Machine			
Note	Machine that represents an Adaptive Autosar Software Stack. **Tags:** atp.Status=draft; atp.recommendedPackage=Machines			
Base	ARElement, ARObject, AtpClassifier, AtpFeature, AtpStructureElement, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
communicationConnector	CommunicationConnector	*	aggr	This aggregation defines the network connection of the machine. **Tags:** atp.Status=draft
defaultApplicationTimeout	EnterExitTimeout	0..1	aggr	This aggration defines a default timeout in the context of a given Machine with respect to the launching and termination of applications. **Tags:** atp.Status=draft
functionGroup	ModeDeclarationGroupPrototype	*	aggr	This aggregation represents the collection of function groups of the enclosing Machine. **Stereotypes:** atpVariation **Tags:** 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
machineModeMachine	ModeDeclarationGroupPrototype	0..1	aggr	Set of MachineStates (Modes) that are defined for the machine. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=preCompileTime
moduleInstantiation	AdaptiveModuleInstantiation	*	aggr	Configuration of Adaptive Autosar module instances that are running on the machine. **Tags:** atp.Status=draft
perStateTimeout	PerStateTimeout	*	aggr	This aggregation represens the definition of per-state-timeouts in the context of the enclosing machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=perStateTimeout; atp.Status=draft
processor	Processor	1..*	aggr	This represents the collection of processors owned by the enclosing machine. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
secureCommunicationDeployment	SecureCommunicationDeployment	*	aggr	Deployment of secure communication protocol configuration settings to crypto module entities. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel; atp.Status=draft
serviceDiscoverConfig	ServiceDiscoveryConfiguration	*	aggr	Set of service discovery configuration settings that are defined on the machine for individual CommunicationConnectors. **Tags:** atp.Status=draft

Table A.40: Machine

Class	NetworkEndpoint			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	The network endpoint defines the network addressing (e.g. IP-Address or MAC multicast address).			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
fullyQualifiedDomainName	String	0..1	attr	Defines the fully qualified domain name (FQDN) e.g. some.example.host.
infrastructureServices	InfrastructureServices	0..1	aggr	Defines the network infrastructure services provided or consumed.
networkEndpointAddress	NetworkEndpointAddress	1..*	aggr	Definition of a Network Address. **Tags:** xml.namePlural=NETWORK-ENDPOINT-ADDRESSES
priority	PositiveInteger	0..1	attr	Priority of this Network-Endpoint.

Table A.41: NetworkEndpoint

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			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName; atp.Status=draft

Attribute	Type	Mul.	Kind	Note

Table A.42: PortInterface

Class	PortInterfaceToDataTypeMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
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=ServiceInterfaceToDataType Mappings			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataTypeMappingSet	DataTypeMappingSet	1..*	ref	This represents the reference to the applicable dataTypemappingSet **Tags:** atp.Status=draft; atp.Status Comment=Reserved for adaptive platform
portInterface	PortInterface	1	ref	This represents the reference to the applicable PortInterface **Tags:** atp.Status=draft; atp.Status Comment=Reserved for adaptive platform

Table A.43: PortInterfaceToDataTypeMapping

Class	ProvidedSomeipServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	This meta-class represents the ability to describe the existence and configuration of a provided service instance in a concrete implementation on top of SOME/IP. **Tags:** atp.Status=draft; atp.recommendedPackage=ServiceInstances			
Base	ARElement, ARObject, AdaptivePlatformServiceInstance, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, ProvidedApService Instance, Referrable, UploadablePackageElement			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
providedEventGroup	SomeipProvidedEventGroup	*	aggr	List of EventGroups that are provided by the Service Instance. **Tags:** atp.Status=draft
sdServerConfig	SomeipSdServerServiceInstanceConfig	0..1	aggr	Server specific configuration settings relevant for the SOME/IP service discovery. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
serviceInstanceId	PositiveInteger	1	attr	Identification number that is used by SOME/IP service discovery to identify the instance of the service.

Table A.44: ProvidedSomeipServiceInstance

Class	ReceiverComSpec (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Receiver-specific communication attributes (RPortPrototype typed by SenderReceiverInterface).			
Base	ARObject, RPortComSpec			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataElement	AutosarDataPrototype	0..1	ref	Data element these attributes belong to.
dataUpdatePeriod	TimeValue	0..1	attr	This attribute defines the period in which the application shall check for updated data. This attribute is used for the configuration of the E2E protection. **Tags:** atp.Status=draft
receiverCapability	ReceiverCapabilityEnum	0..1	attr	This attribute represents the expressed capability of the receiver. The receiver may decide to claim that existing resources of a ServiceInterface are expressly not used by this specific receiver. The conceptual background of this claim may be driven by security, safety, etc. **Tags:** atp.Status=draft
transformationComSpecProps	TransformationComSpecProps	*	aggr	This references the TransformationComSpecProps which define port-specific configuration for data transformation.

Table A.45: ReceiverComSpec

Class	Referrable (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	ARObject			
Attribute	**Type**	**Mul.**	**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

Attribute	Type	Mul.	Kind	Note
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.46: Referrable

Class	RequiredSomeipServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	This meta-class represents the ability to describe the existence and configuration of a required service instance in a concrete implementation on top of SOME/IP. **Tags:** atp.Status=draft; atp.recommendedPackage=ServiceInstances			
Base	ARElement, ARObject, AdaptivePlatformServiceInstance, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, RequiredApServiceInstance, UploadablePackageElement			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
requiredEventGroup	SomeipRequiredEventGroup	*	aggr	List of EventGroups that are used by the RequiredServiceInstance. **Tags:** atp.Status=draft
requiredServiceInstanceId	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID.
requiredServiceVersion	SomeipServiceInterfaceVersion	0..1	aggr	This element is used to configure for which version (major version/minor version) of the SomeIp Service the Service Discovery will search. **Tags:** atp.Status=draft
sdClientConfig	SomeipSdClientServiceInstanceConfig	0..1	aggr	Client specific configuration settings relevant for the SOME/IP service discovery. **Tags:** atp.Status=draft

Table A.47: RequiredSomeipServiceInstance

Class	SecOcSecureComProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::SecureCommunication			
Note	Configuration of AUTOSAR SecOC. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, SecureComProps			
Attribute	**Type**	**Mul.**	**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.

Attribute	Type	Mul.	Kind	Note
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.48: SecOcSecureComProps

Class	SecureComProps (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::SecureCommunication			
Note	This meta-class defines a communication security protocol and its configuration settings. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.49: SecureComProps

Class	SenderComSpec (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a sender port (PPortPrototype typed by SenderReceiverInterface).			
Base	ARObject, PPortComSpec			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
compositeNetworkRepresentation	CompositeNetworkRepresentation	*	aggr	This represents a CompositeNetworkRepresentation defined in the context of a SenderComSpec.
dataElement	AutosarDataPrototype	0..1	ref	Data element these quality of service attributes apply to.
dataUpdatePeriod	TimeValue	0..1	attr	This attribute describes the period in which the applications are assumed to transmit E2E-protected messages. The middleware does not use this attribute at all. **Tags:** atp.Status=draft
networkRepresentation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the dataElement is mapped to a communication bus.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
senderCapability	SenderCapabilityEnum	0..1	attr	This attribute represents the expressed capability of the sender. The sender may decide to claim that existing resources of a ServiceInterface are expressly not used by this specific sender. The conceptual background of this claim may be driven by security, safety, etc. **Tags:** atp.Status=draft
transmissionAcknowledge	TransmissionAcknowledgementRequest	0..1	aggr	Requested transmission acknowledgement for data element.
usesEndToEndProtection	Boolean	1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table A.50: SenderComSpec

Class	**ServiceInstanceToMachineMapping (abstract)**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstanceMapping			
Note	This meta-class represents the ability to map a AdaptivePlatformServiceInstance to a CommunicationConnector of a Machine. **Tags:** atp.Status=draft			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
communicationConnector	CommunicationConnector	0..1	ref	Reference to the Machine to which the ServiceInstance is mapped. **Tags:** atp.Status=draft
serviceInstance	AdaptivePlatformServiceInstance	0..1	ref	Reference to a ServiceInstance that is mapped to the Machine. **Tags:** atp.Status=draft

Table A.51: ServiceInstanceToMachineMapping

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, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Port Interface, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
event	VariableDataPr ototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
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
method	ClientServerOp eration	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
optionalEle ment	ServiceInterface SubElement	*	aggr	This aggregation represents the collectionof optional elements within the scope of the enclosing ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel; atp.Status=draft vh.latestBindingTime=blueprintDerivationTime
possibleErr or	ApplicationError	*	aggr	This represents the collection of ApplicationErrors defined in the context of the enclosing ServiceInterface. **Tags:** atp.Status=draft

Table A.52: ServiceInterface

Class	**ServiceInterfaceDeployment (abstract)**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterface Deployment			
Note	Middleware transport layer specific configuration settings for the ServiceInterface and all contained ServiceInterface elements. **Tags:** atp.Status=draft			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadablePackageElement			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
eventDeployment	ServiceEventDeployment	*	aggr	Middleware transport layer specific configuration settings for an Event that is defined in the ServiceInterface. **Tags:** atp.Status=draft
fieldDeployment	ServiceFieldDeployment	*	aggr	Middleware transport layer specific configuration settings for a Field that is defined in the ServiceInterface. **Tags:** atp.Status=draft
methodDeployment	ServiceMethodDeployment	*	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.53: ServiceInterfaceDeployment

Class	**ServiceInterfaceElementSecureComConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::SecureCommunication			
Note	This element allows to secure the communication of the referenced ServiceInterface element. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataId	PositiveInteger	0..1	attr	This attribute defines a unique numerical identifier for the referenced ServiceInterface element.
event	ServiceEventDeployment	0..1	ref	Reference to an event that is protected by a security protocol. **Tags:** atp.Status=draft
fieldNotifier	ServiceFieldDeployment	0..1	ref	Reference to a field notifier that is protected by a security protocol. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
freshnessValueId	PositiveInteger	0..1	attr	This attribute defines the Id of the Freshness Value.
getterCall	ServiceFieldDeployment	0..1	ref	Reference to a field getter call message that is protected by a security protocol. **Tags:** atp.Status=draft
getterReturn	ServiceFieldDeployment	0..1	ref	Reference to a field getter return message that is protected by a security protocol. **Tags:** atp.Status=draft
methodCall	ServiceMethodDeployment	0..1	ref	Reference to a method call message that is protected by a security protocol. **Tags:** atp.Status=draft
methodReturn	ServiceMethodDeployment	0..1	ref	Reference to a method return message that is protected by a security protocol. **Tags:** atp.Status=draft
secureComProps	SecureComProps	0..1	ref	Reference to the communication security protocol and its configuration settings that will provide communication security for the referenced ServiceInterfaceElement that is exchanged between a ProvidedServiceInstance and one or several RequiredServiceInstances. **Tags:** atp.Status=draft
setterCall	ServiceFieldDeployment	0..1	ref	Reference to a field setter call message that is protected by a security protocol. **Tags:** atp.Status=draft
setterReturn	ServiceFieldDeployment	0..1	ref	Reference to a field setter return message that is protected by a security protocol. **Tags:** atp.Status=draft

Table A.54: ServiceInterfaceElementSecureComConfig

Class	ServiceMethodDeployment (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterfaceDeployment			
Note	This abstract meta-class represents the ability to specify a deployment of a Method to a middleware transport layer. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**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

Attribute	Type	Mul.	Kind	Note

Table A.55: ServiceMethodDeployment

Class	SomeipDataPrototypeTransformationProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::TransformationConfiguration			
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=SomeipDataPrototype TransformationPropss			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataPrototype	CompositionDataPrototypeRef	*	aggr	Collection of DataPrototypes for which the settings in SomeipDataPrototypeTransformationProps are valid. For reuse reasons the SomeipDataPrototypeTransformationProps 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 networkRepresentation is not provided then the baseType of the ImplementationDataType shall be used for the serialization/deserialization. **Tags:** atp.Status=draft
someipTransformationProps	ApSomeipTransformationProps	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.56: SomeipDataPrototypeTransformationProps

Class	SomeipEventDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterface Deployment			
Note	SOME/IP configuration settings for an Event. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceEvent Deployment			
Attribute	**Type**	**Mul.**	**Kind**	**Note**

Attribute	Type	Mul.	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.
transportProtocol	TransportLayerProtocolEnum	1	attr	This attribute defines over which Transport Layer Protocol this event is intended to be sent.

Table A.57: SomeipEventDeployment

Class	SomeipEventGroup			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterfaceDeployment			
Note	Grouping of events and notification events inside a ServiceInterface in order to allow subscriptions. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
event	SomeipEventDeployment	*	ref	Reference to an event that is part of the EventGroup. **Tags:** atp.Status=draft
eventGroupId	PositiveInteger	1	attr	Unique Identifier that identifies the EventGroup in SOME/IP. This Identifier is sent as Eventgroup ID in SOME/IP Service Discovery messages.

Table A.58: SomeipEventGroup

Class	**SomeipFieldDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterface Deployment			
Note	SOME/IP configuration settings for a Field. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceFieldDeployment			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
get	SomeipMethod Deployment	0..1	aggr	This aggregation represents the setting of the get method. **Tags:** atp.Status=draft
notifier	SomeipEventDe ployment	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.59: SomeipFieldDeployment

Class	**SomeipMethodDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterface Deployment			
Note	SOME/IP configuration settings for a Method. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceMethod Deployment			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
maximumS egmentLen gthReques t	PositiveInteger	0..1	attr	This attribute describes the length in bytes of one SOME/IP segment into which the Method Call Message will be divided. This length field includes 8 bytes for the Request ID, Protocol Version, Interface Version, Message Type and Return Code and 4 additional SOME/IP TP bytes. If this attribute is set to a value and the data length is larger than maximumSegmentLengthRequest then the corresponding SOME/IP message will be segmented into smaller parts that are transmitted over the network.

Attribute	Type	Mul.	Kind	Note
maximumSegmentLengthResponse	PositiveInteger	0..1	attr	This attribute describes the length in bytes of one SOME/IP segment into which the Method Return Message will be divided. This length field includes 8 bytes for the Request ID, Protocol Version, Interface Version, Message Type and Return Code and 4 additional SOME/IP TP bytes. If this attribute is set to a value and the data length is larger than maximumSegmentLengthResponse then the corresponding SOME/IP message will be segmented into smaller parts that are transmitted over the network.
methodId	PositiveInteger	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	TransportLayerProtocolEnum	1	attr	This attribute defines over which Transport Layer Protocol this method is intended to be sent.

Table A.60: SomeipMethodDeployment

Class	SomeipProvidedEventGroup			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	The meta-class represents the ability to configure ServiceInstance related communication settings on the provided side for each EventGroup separately. **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
eventGroup	SomeipEventGroup	0..1	ref	Reference to the SomeipEventGroup in the System Manifest for which the ServiceInstance related EventGroup settings are valid. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
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 server 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.
sdServerEventConfig	SomeipSdServerEventTimingConfig	0..1	aggr	Server Timing configuration settings that are EventGroup specific. **Tags:** atp.Status=draft

Table A.61: SomeipProvidedEventGroup

Class	SomeipRequiredEventGroup			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	The meta-class represents the ability to configure ServiceInstance related communication settings on the required side for each EventGroup separately. **Tags:** atp.Status=draft			
Base	ARObject, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
eventGroup	SomeipEventGroup	0..1	ref	Reference to the SomeipEventGroup in the System Manifest for which the ServiceInstance related EventGroup settings are valid. **Tags:** atp.Status=draft
sdClientEventTimingConfig	SomeipSdClientEventGroupTimingConfig	0..1	aggr	Client Timing configuration settings that are EventGroup specific. **Tags:** atp.Status=draft

Table A.62: SomeipRequiredEventGroup

Class	SomeipSdClientEventGroupTimingConfig			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
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.Status=draft			
Base	ARObject			
Attribute	**Type**	**Mul.**	**Kind**	**Note**

Attribute	*Type*	*Mul.*	*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). **Tags:** atp.Status=draft
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.63: SomeipSdClientEventGroupTimingConfig

Class	**SomeipSdClientServiceInstanceConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	Client specific settings that are relevant for the configuration of SOME/IP Service-Discovery. **Tags:** atp.Status=draft			
Base	ARObject			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
capabilityRecord	TagWithOptionalValue	*	aggr	A sequence of records to store arbitrary name/value pairs conveying additional information about the named service. **Tags:** atp.Status=draft
initialFindBehavior	InitialSdDelayConfig	0..1	aggr	Controls initial find behavior of clients. **Tags:** atp.Status=draft
serviceFindTimeToLive	PositiveInteger	1	attr	This attribute represents the ability to define the time in seconds the service find is valid.

Table A.64: SomeipSdClientServiceInstanceConfig

Class	**SomeipSdServerServiceInstanceConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	Server specific settings that are relevant for the configuration of SOME/IP Service-Discovery. **Tags:** atp.Status=draft			
Base	ARObject			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
capabilityRecord	TagWithOptionalValue	*	aggr	A sequence of records to store arbitrary name/value pairs conveying additional information about the named service. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
initialOfferBehavior	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).
requestResponseDelay	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
serviceOfferTimeToLive	PositiveInteger	1	attr	Defines the time in seconds the service offer is valid.

Table A.65: SomeipSdServerServiceInstanceConfig

Class	SomeipServiceInstanceToMachineMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstanceMapping			
Note	This meta-class allows to map SomeipServiceInstances to a CommunicationConnector of a Machine. In this step the network configuration (IP Address, Transport Protocol, Port Number) for the ServiceInstance is defined. **Tags:** atp.Status=draft; atp.recommendedPackage=ServiceInstanceToMachine Mappings			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, ServiceInstanceToMachineMapping, Uploadable PackageElement			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
eventMulticastUdpPort	PositiveInteger	0..1	attr	UdpPort configuration that is used for Event communication in the IP-Multicast case. 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 mulicastThreshold of the corresponding ProvidedEventGroupInSomeipServiceInstance is exceeded.
ipv4MulticastIpAddress	Ip4AddressString	0..1	attr	Multicast IPv4 Address that is transmitted in the EventGroupSubscribeAck message for all available EventGroups that are available in the ProvidedSomeipServiceInstance.
ipv6MulticastIpAddress	Ip6AddressString	0..1	attr	Multicast IPv6 Address that is transmitted in the EventGroupSubscribeAck message for all available EventGroups that are available in the ProvidedSomeipServiceInstance.

Attribute	Type	Mul.	Kind	Note
tcpPort	PositiveInteger	0..1	attr	TcpPort configuration that is used for Method and Event communication in IP-Unicast case. SOME/IP Service Discovery: PortNumber that is sent in the SD-Offer Message to client (answer on SD-find) or clients (SD-offer). Method: This is the destination-port where the server accepts the method call messages (from the clients). This is the source-port where the server sends the method response messages (to the client). Event: This is the event source-port where the server sends the event messages to the subscribed clients in IP-Unicast case.
udpPort	PositiveInteger	0..1	attr	UdpPort configuration that is used for Method and Event communication in IP-Unicast case. SOME/IP Service Discovery: PortNumber that is sent in the SD-Offer Message to client (answer on SD-find) or clients (SD-offer). Method: This is the destination-port where the server accepts the method call messages (from the clients). This is the source-port where the server sends the method response messages (to the client). Event: This is the event source-port where the server sends the event messages to the subscribed clients in IP-Unicast case.

Table A.66: SomeipServiceInstanceToMachineMapping

Class	SomeipServiceInterfaceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInterface Deployment			
Note	SOME/IP configuration settings for a ServiceInterface. **Tags:** atp.Status=draft; atp.recommendedPackage=ServiceInterfaceDeployments			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, ServiceInterfaceDeployment, UploadablePackage Element			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
eventGroup	SomeipEventGroup	*	aggr	SOME/IP EventGroups that are defined within the SOME/IP ServiceClass. **Tags:** atp.Status=draft

Attribute	Type	Mul.	Kind	Note
serviceInterfaceId	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.
serviceInterfaceVersion	SomeipServiceInterfaceVersion	1	aggr	The SOME/IP major and minor Version of the Service. **Tags:** atp.Status=draft

Table A.67: SomeipServiceInterfaceDeployment

Class	SomeipServiceInterfaceVersion			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance			
Note	This meta-class represents the ability to describe a version of a SOME/IP ServiceInterface. **Tags:** atp.Status=draft			
Base	ARObject			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
majorVersion	AnyVersionString	1	attr	Major Version of the ServiceInterface. Value can be set to a number that represents the Major Version of the searched service or to ANY.
minorVersion	AnyVersionString	1	attr	Minor Version of the ServiceInterface. Value can be set to a number that represents the Minor Version of the searched service or to ANY.

Table A.68: SomeipServiceInterfaceVersion

Class	SwBaseType			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, BaseType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.69: SwBaseType

Class	≪atpVariation≫ **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNativeTypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	ARObject			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
additionalNativeTypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50

Attribute	*Type*	*Mul.*	*Kind*	*Note*
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
implementationDataType	ImplementationDataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see 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 **Tags:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30

Attribute	*Type*	*Mul.*	*Kind*	*Note*
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 memoryAllocationKeywordPolicy of the referenced SwAddrMethod. **Tags:** xml.sequenceOffset=33
swBitRepresentation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxisSet	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparisonVariable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170; xml.typeElement=false
swDataDependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220; xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230

Attribute	Type	Mul.	Kind	Note
swIntendedResolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolationMethod	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTargetProps	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecordLayout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290
swRefreshTiming	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300

Attribute	Type	Mul.	Kind	Note
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBlockSize	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisDataType	ApplicationPrimitiveDataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table A.70: 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**	**Mul.**	**Kind**	**Note**
arraySizeSemantics	ArraySizeSemanticsEnum	1	attr	This attribute controls the semantics of the arraysize for the array representing the string in an ImplementationDataType. It is there to support a safe conversion between ApplicationDatatype and ImplementationDatatype, even for variable length strings as required e.g. for Support of SAE J1939.
baseType	SwBaseType	0..1	ref	This is the base type of one character in the string. In particular this baseType denotes the intended encoding of the characters in the string on level of ApplicationDataType. **Tags:** xml.sequenceOffset=30

Attribute	*Type*	*Mul.*	*Kind*	*Note*
swFillCharacter	Integer	0..1	attr	Filler character for text parameter to pad up to the maximum length swMaxTextSize. The value will be interpreted according to the encoding specified in the associated base type of the data object, e.g. 0x30 (hex) represents the ASCII character zero as filler character and 0 (dec) represents an end of string as filler character. The usage of the fill character depends on the arraySizeSemantics. **Tags:** xml.sequenceOffset=40
swMaxTextSize	Integer	1	attr	Specifies the maximum text size in characters. Note the size in bytes depends on the encoding in the corresponding baseType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Table A.71: SwTextProps

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to attach with the symbol attribute a symbolic name that is conform to C language requirements to another meta-class, e.g. AtomicSwComponentType, that is a potential subject to a name clash on the level of RTE source code.			
Base	ARObject, ImplementationProps, Referrable			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table A.72: SymbolProps

Class	**TlsSecureComProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::SecureCommunication			
Note	Configuration of the Transport Layer Security protocol (TLS). **Tags:** atp.Status=draft			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable, SecureComProps			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
supportedCipherSuite	TlsCipherSuite	*	aggr	Collection of supported cipher suites that are used to negotiate the security settings for a network connection. **Tags:** atp.Status=draft

Table A.73: TlsSecureComProps

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, Identifiable, MultilanguageReferrable, Referrable			
Attribute	*Type*	*Mul.*	*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
transformationProps	TransformationProps	0..1	ref	This represents the reference to the applicable Serialization properties. **Tags:** atp.Status=draft

Table A.74: TransformationPropsToServiceInterfaceElementMapping

Class	**TransformationPropsToServiceInterfaceElementMappingSet**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::TransformationConfiguration			
Note	Collection of TransformationPropsToServiceInterfaceElementMappings. **Tags:** atp.Status=draft; atp.recommendedPackage=TransformationPropsToServiceInterfaceMappingSets			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
mapping	TransformationPropsToServiceInterfaceElementMapping	*	aggr	Mapping that assigns serialization properties to elements of a ServiceInterface. **Tags:** atp.Status=draft

Table A.75: TransformationPropsToServiceInterfaceElementMappingSet

Enumeration	**TransportLayerProtocolEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::Deployment::ServiceInstance
Note	This enumeration allows to choose a TCP/IP transport layer protocol. **Tags:** atp.Status=draft
Literal	*Description*

tcp	Transmission control protocol **Tags:** atp.EnumerationValue=1
udp	User datagram protocol **Tags:** atp.EnumerationValue=0

Table A.76: TransportLayerProtocolEnum

Class	**VariableDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided. In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes.			
Base	ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table A.77: 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
[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

Number	Heading
[SWS_CM_00129]	Ensuring existence of SetHandler
[SWS_CM_00132]	Existence of getter method
[SWS_CM_00133]	Existence of the set method
[SWS_CM_00182]	Event Receive Handler call serialization
[SWS_CM_00183]	Disable service event trigger
[SWS_CM_00252]	
[SWS_CM_00253]	
[SWS_CM_00254]	
[SWS_CM_00255]	
[SWS_CM_00256]	
[SWS_CM_00257]	
[SWS_CM_00258]	
[SWS_CM_00259]	
[SWS_CM_00260]	
[SWS_CM_00262]	
[SWS_CM_00263]	
[SWS_CM_00264]	
[SWS_CM_00265]	
[SWS_CM_00266]	FilterFunction for incoming event filtering
[SWS_CM_00427]	String Data Type with `baseTypeSize` of 16
[SWS_CM_00428]	Element specification typed by String Data Type with `baseTypeSize` of 16
[SWS_CM_01031]	Service fields namespace
[SWS_CM_10268]	
[SWS_CM_10269]	
[SWS_CM_10270]	
[SWS_CM_10271]	
[SWS_CM_10272]	
[SWS_CM_10273]	
[SWS_CM_10274]	
[SWS_CM_10275]	
[SWS_CM_10276]	
[SWS_CM_10277]	
[SWS_CM_10278]	
[SWS_CM_10279]	
[SWS_CM_10280]	
[SWS_CM_10281]	
[SWS_CM_10282]	
[SWS_CM_10283]	
[SWS_CM_10284]	
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10286]	Encoding mismatch in input configurations

Number	Heading
[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 mesages
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10318]	Invoke the notification function
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10320]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10321]	Source of a SOME/IP event message
[SWS_CM_10322]	Destination of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10324]	Checks for a received SOME/IP event message
[SWS_CM_10325]	Identifying the right event
[SWS_CM_10326]	Deserializing the payload

Number	Heading
[SWS_CM_10327]	Store the received event data
[SWS_CM_10328]	Invoke receive handler
[SWS_CM_10329]	Conditions for sending of a SOME/IP request message
[SWS_CM_10330]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10331]	Source of a SOME/IP request message
[SWS_CM_10332]	Destination of a SOME/IP request message
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10335]	Identifying the right method
[SWS_CM_10336]	Deserializing the payload
[SWS_CM_10337]	Store the received method data
[SWS_CM_10338]	Invoke the registered set/get handlers - event driven
[SWS_CM_10339]	Invoke the registered set/get handlers - polling
[SWS_CM_10340]	Conditions for sending of a SOME/IP response message
[SWS_CM_10341]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_10342]	Source of a SOME/IP response message
[SWS_CM_10343]	Destination of a SOME/IP response message
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10345]	Checks for a received SOME/IP response message
[SWS_CM_10346]	Identifying the right method
[SWS_CM_10347]	Discarding orphaned responses
[SWS_CM_10348]	Deserializing the payload
[SWS_CM_10349]	Making the Future ready
[SWS_CM_10350]	Invoke the notification function
[SWS_CM_10351]	Service application errors
[SWS_CM_10352]	Definition of `ServiceNotAvailableException`
[SWS_CM_10353]	Use of `ServiceNotAvailableException`
[SWS_CM_10354]	Definition of `ApplicationErrorException`
[SWS_CM_10355]	Use of `ApplicationErrorException`
[SWS_CM_10356]	Definition of sub-classes of `ApplicationErrorException`
[SWS_CM_10357]	Distinguishing errors from normal responses
[SWS_CM_10358]	Identifying the right application error
[SWS_CM_10359]	Deserializing the payload - error response mesages
[SWS_CM_10361]	
[SWS_CM_10362]	Raising checked exceptions for application errors
[SWS_CM_10370]	Data Type definitions for Application Errors in Common header file
[SWS_CM_10371]	Context of thrown checked exceptions
[SWS_CM_11262]	
[SWS_CM_11263]	
[SWS_CM_90101]	Secure channel creation

Number	Heading
[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]	
[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::e2e:state_machine::E2E check status
[SWS_CM_90422]	ara::e2e: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
[SWS_CM_00307]	Sample Container
[SWS_CM_00308]	Sample Allocatee Pointer
[SWS_CM_00309]	Event Receive Handler
[SWS_CM_00310]	Subscription State
[SWS_CM_00312]	Handle Type Class
[SWS_CM_00346]	`Promise::set_value`, forwarding reference version
[SWS_CM_00406]	String Data Type with `baseTypeSize` of 8
[SWS_CM_00409]	Associative Map Data Type
[SWS_CM_00420]	Element specification typed by String Data Type with `baseTypeSize` of 8
[SWS_CM_01010]	Service Identifier and Service Version Classes
[SWS_CM_01016]	Data Type definitions for AUTOSAR Data Types in Common header file
[SWS_CM_01019]	Data Type declarations in Types header file
[SWS_CM_10017]	
[SWS_CM_10034]	
[SWS_CM_10059]	
[SWS_CM_10242]	UTF-8 Strings
[SWS_CM_10243]	UTF-16 Strings
[SWS_CM_10245]	Serialization of strings
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10252]	
[SWS_CM_10253]	
[SWS_CM_10256]	
[SWS_CM_10257]	
[SWS_CM_10258]	
[SWS_CM_10260]	

Number	Heading
[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