| Document Title | Specification of SOME/IP Transformer |
|---|---|
| **Document Category** | Software specification (SWS) |
| **Document UID** | 660 |
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R26-11 |

**Document Change History**

| Date | Release | Changed by | Description |
|---|---|---|---|
| 2026-11-26 | R26-11 | AUTOSAR Release Management | • Added security-event reporting and context-data definitions for SOME/IP processing errors, along with new rules for string length-field insertion, length-field typing, serialized string payload sizing, and maximum variable-array elements.<br>• Updated fixed-length string serialization and deserialization behavior, including BOM handling, length calculation, padding, and configuration-dependent defaults.<br>• Refined development-error classification, security-event context definitions, and imported datatype/API specifications for improved consistency with related modules. |
| 2025-11-27 | R25-11 | AUTOSAR Release Management | • Reverted the handling of receiving less data than expected and substitution of missing elements with default values during deserialization<br>• Marked SOMEIPXF_E_NOT_READY , SOMEIPXF_E_NOT_REACHABLE, SOMEIPXF_E_TIMEOUT as deprecated<br>• Updated the default behavior/values for (de-)serialization parameters<br>• Added a reference to new error code E_SER_PAYLOAD_LENGTH_ EXCEEDED that is issued when array length is greater than expected ▽ |

▽

| | | | |
|---|---|---|---|
| | | | • Clarified the deserialization for duplicate members and invalid wire type<br>• Updated the length field requirements for fixed length string<br>• Added requirements for evaluating return code in case of autonomous error response of a Client-Server operation<br>• Editorial Changes and bug fixes |
| 2024-11-27 | R24-11 | AUTOSAR Release Management | • Clarified handling of Message Types RESPONSE(0x80) and ERROR(0x81)<br>• Clarified handling of UTF-8 and UTF-16<br>• New section 'De-serialization of Parameters and Data Structures'<br>• Fix of Uptraces and Editorial Changes |
| 2023-11-23 | R23-11 | AUTOSAR Release Management | • Restrict SOME/IP session handling<br>• Added information about use of maximum number of array elements<br>• Removed chapter "Header file structure"<br>• Clarification on byte order for SOME/IP Header fields, additional fields in the payload and parameters in payload |
| 2022-11-24 | R22-11 | AUTOSAR Release Management | • implementsSOMEIPStringHandling SWS_SomeIpXf_00239 set to OBSOLETE)<br>• Reworked [SWS_SomeIpXf_00054] with regards to UTF-8<br>• Clarified byte order of length fields within SOME/IP payload<br>• Extended [SWS_SomeIpXf_00300] to support uint64<br>• Distinguished in [SWS_SomeIpXf_00200] use of error messages with Message ID ERROR not clearly distinguished from use of autonomous error responses<br>• Bugs resolved within [SWS_SomeIpXf_00244] and [SWS_SomeIpXf_00303] |

| | | | |
|---|---|---|---|
| | | | • Resolved mismatch in the size of Method-ID on SWS_SOMEIPTransformer and PRS_SOMEIPProtocol. Removed chapters 7.2.3.1 "Message ID [32 bit]", 7.2.3.2 "Length [32 bit], 7.1 "Definition of Identifiers" and 7.4 "Reserved and special identifiers for SOME/IP and SOME/IP-SD"<br>• Editorial Changes |
| 2021-11-25 | R21-11 | AUTOSAR Release Management | • Clarification on network representation<br>• SOME/IP Header encoded in network byte order<br>• Clarification on SOMEIPLegacyStringSerialization<br>• Optional method arguments not supported<br>• Clarification on Interface Version<br>• Clarification on processing order of header fields in AUTOSAR CP<br>• Removed SOMEIPXF_E_UNKNOWN_SERVICE and SOMEIPXF_E_UNKNOWN_METHOD<br>• Introduction on External Trigger Events<br>• Clarification on ISignal length of external trigger event<br>• Editorial Changes |
| 2020-11-30 | R20-11 | AUTOSAR Release Management | • Added call/response context to Client Server requirements<br>• Constraint added for data type of length field of variable Strings<br>• Added E_E2E Error to Table 7.11: Return Codes<br>• Requirement added in case unavailability of optional member in the received serialized byte stream<br>• Reworked E2E communication protection for methods |

△

| | | | |
|---|---|---|---|
| | | | △<br>• sizeOfStringLengthField introduced for the size of the length field for dynamic length strings<br>• sizeOfArrayLengthField introduced for the size of the length field for variable size arrays<br>• Fixed design issues with E2E communication protection for methods<br>• Editorial Changes |
| 2019-11-28 | R19-11 | AUTOSAR Release Management | • Extended Serialization for Data Structures in SOME/IP with tag/length/value encoding set to valid<br>• Removed *_ACK message types<br>• replaced implementsSOMEIPStringHandling (in class SOMEIPTransformationSignalProps) with implementsLegacyStringSerialization<br>• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation<br>• Changed Document Status from Final to published |
| 2018-10-31 | 4.4.0 | AUTOSAR Release Management | • Checking for length of received dynamic length strings<br>• Extended Serialization for Data Structures in SOME/IP with tag/length/value encoding<br>• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation |
| 2017-12-08 | 4.3.1 | AUTOSAR Release Management | • Bugfixes in serialization of strings and data with variable size<br>• Signatures improved<br>• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation |

▽

△

| | | | |
|---|---|---|---|
| 2016-11-30 | 4.3.0 | AUTOSAR Release Management | • Sizes of length fields can be configured independently from each other<br>• Support of union data types<br>• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation |
| 2015-07-31 | 4.2.2 | AUTOSAR Release Management | • Size of length fields is configurable<br>• External trigger events are communicated as fire-and-forget methods<br>• Autonomous error reactions of SOME/IP transformer<br>• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation |
| 2014-10-31 | 4.2.1 | AUTOSAR Release Management | • Initial Release |

**Disclaimer**

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

# Table of Contents

1 Introduction and functional overview 10
2 Acronyms and Abbreviations 11
3 Related documentation 12
3.1 Input documents . . . 12
3.2 Related standards and norms . . . 13
3.3 Related specification . . . 13
4 Constraints and assumptions 14
4.1 Limitations . . . 14
4.2 Applicability to car domains . . . 14
5 Dependencies to other modules 15
5.1 File structure . . . 15
5.1.1 Code file structure . . . 15
6 Requirements Tracing 16
7 Functional specification 22
7.1 Specification of the SOME/IP on-wire format . . . 25
7.1.1 Message Length Limitations . . . 25
7.1.2 Endianness . . . 25
7.1.3 Message format . . . 26
7.1.3.1 Request ID [32 bit] . . . 27
7.1.3.2 Protocol Version [8 bit] . . . 28
7.1.3.3 Interface Version [8 bit] . . . 29
7.1.3.4 Message Type [8 bit] . . . 29
7.1.3.5 Return Code [8 bit] . . . 30
7.1.3.6 Payload [variable size] . . . 30
7.1.4 Serialization of Parameters and Data Structures . . . 31
7.1.4.1 Basic Datatypes . . . 33
7.1.4.2 Structured Datatypes (structs) . . . 33
7.1.4.3 Structured Datatypes and Arguments with Identifier and optional Members . . . 36
7.1.4.4 Strings . . . 42
7.1.4.5 Arrays (fixed length) . . . 49
7.1.4.6 Optional Parameters / Optional Elements . . . 52
7.1.4.7 Dynamic Length Arrays / Variable Size Arrays . . . 52
7.1.4.8 Bitfield . . . 56
7.1.4.9 Union / Variant . . . 56
7.1.5 De-serialization of Parameters and Data Structures . . . 60
7.1.5.1 Structured Datatypes (structs) . . . 61

7.1.5.2 Structured Datatypes and Arguments with Identifier and optional Members . . . 61
7.1.5.3 Strings . . . 62
7.1.5.4 Arrays (fixed length) . . . 64
7.1.5.5 Dynamic Length Arrays / Variable Size Arrays . . . 64
7.1.5.6 Bitfield . . . 64
7.1.5.7 Union / Variant . . . 64
7.2 Protocol specification . . . 64
7.2.1 Client/Server Communication . . . 65
7.2.2 Sender/Receiver Communication . . . 68
7.2.3 External Trigger Events . . . 69
7.2.4 Error Handling . . . 70
7.2.4.1 Return Code . . . 71
7.2.4.2 Communication Errors and Handling of Communication Errors . 72
7.2.5 Security Events . . . 74
7.2.5.1 Security Event Overview . . . 74
7.3 Error Classification . . . 75
7.3.1 Development Errors . . . 75
7.3.2 Runtime Errors . . . 76
7.3.3 Production Errors . . . 76
8 API specification 77
8.1 Imported types . . . 77
8.2 Type definitions . . . 77
8.3 Function definitions . . . 77
8.3.1 SomeIpXf_ExtractProtocolHeaderFields . . . 78
8.3.2 SomeIpXf_<transformerId> . . . 81
8.3.3 SomeIpXf_Inv_<transformerId> . . . 86
8.3.4 SomeIpXf_Init . . . 92
8.3.5 SomeIpXf_DeInit . . . 92
8.4 Scheduled functions . . . 93
8.5 Expected interfaces . . . 93
8.5.1 Mandatory Interfaces . . . 93
8.5.2 Optional Interfaces . . . 93
9 Sequence diagrams 94
10 Configuration specification 95
A Not applicable requirements 96
B Change History 97
B.1 Change History R26-11 . . . 97
B.1.1 Added Specification Items in R26-11 . . . 97
B.1.2 Changed Specification Items in R26-11 . . . 97
B.1.3 Deleted Specification Items in R26-11 . . . 97

B.1.4 Added Constraints in R26-11 . . . 97
B.1.5 Changed Constraints in R26-11 . . . 97
B.1.6 Deleted Constraints in R26-11 . . . 97
B.2 Change History R25-11 . . . 98
B.2.1 Added Specification Items in R25-11 . . . 98
B.2.2 Changed Specification Items in R25-11 . . . 98
B.2.3 Deleted Specification Items in R25-11 . . . 98
B.2.4 Added Constraints in R25-11 . . . 98
B.2.5 Changed Constraints in R25-11 . . . 98
B.2.6 Deleted Constraints in R25-11 . . . 98
B.3 Change History R24-11 . . . 98
B.3.1 Added Specification Items in R24-11 . . . 98
B.3.2 Changed Specification Items in R24-11 . . . 99
B.3.3 Deleted Specification Items in R24-11 . . . 99
B.3.4 Added Constraints in R24-11 . . . 99
B.3.5 Changed Constraints in R24-11 . . . 99
B.3.6 Deleted Constraints in R24-11 . . . 99
B.4 Change History R23-11 . . . 99
B.4.1 Added Specification Items in R23-11 . . . 99
B.4.2 Changed Specification Items in R23-11 . . . 99
B.4.3 Deleted Specification Items in R23-11 . . . 99
B.4.4 Added Constraints in R23-11 . . . 100
B.4.5 Changed Constraints in R23-11 . . . 100
B.4.6 Deleted Constraints in R23-11 . . . 100
B.5 Change History R22-11 . . . 100
B.5.1 Added Specification Items in R22-11 . . . 100
B.5.2 Changed Specification Items in R22-11 . . . 100
B.5.3 Deleted Specification Items in R22-11 . . . 100
C Referenced Meta Classes 101
D Features of SOME/IP not supported by AUTOSAR SOME/IP transformer 124
E Examples 125
E.1 Serialization of a Client/Server Operation . . . 125
E.1.1 Client . . . 126
E.1.2 Server . . . 127

# 1 Introduction and functional overview

This document specifies the **Scalable service-Oriented MiddlewarE over IP (SOME/IP) Transformer**. This is a transformer which linearizes data with the SOME/IP on-the-wire format and specifies an automotive/embedded mechanism for Client/Server communication.

The only valid abbreviation is SOME/IP. Other abbreviations (e.g. Some/IP) are wrong and shall not be used.

The basic motivation to specify "yet another Client/Server and Sender/Receiver mechanism" instead of using an existing infrastructure/technology is the goal to have a technology that:

- Fulfills the hard requirements regarding resource consumption in an embedded world
- Is compatible through as many use-cases and communication partners as possible
- Provides the features required by automotive use-cases
- Is scalable from tiny to large platforms
- Can be implemented on different operating system (i.e. AUTOSAR, GENIVI, and OSEK) and even embedded devices without operating system

# 2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the SOME/IP Transformer that are not included in the [1, AUTOSAR glossary].

| Abbreviation / Acronym: | Description: |
|---|---|
| Client-Service-Instance-Entry | The configuration and required data of a service instance another ECU offers shall be called Client-Service-Instance-Entry at the ECU using this service (Client). |
| Field | a field represents a status and thus has a valid value at all times on which getter, setter and notifier act upon. |
| Finding a service instance | to send a SOME/IP-SD message in order to find a needed service instance. |
| Getter | a Request/Response call that allows read access to a field. |
| Method | a method, procedure, function, or subroutine that is called/invoked. |
| Notifier | sends out event message with a new value on change of the value of the field. |
| Request | a message of the client to the server invoking a method. |
| Response | a message of the server to the client transporting results of a method invocation. |
| SD | Service Discovery(see[2]) |
| Service | a logical combination of zero or more methods, zero or more events, and zero or more fields. |
| Service Instance | software implementation of the service interface, which can exist more than once in the vehicle and more than once on an ECU. |
| Service Interface | the formal specification of the service including its methods, events, and fields. |
| Setter | a Request/Response call that allows write access to a field. |
| SOME/IP | Scalable service-Oriented MiddlewarE over IP |

**Table 2.1: Acronyms and Abbreviations**

# 3 Related documentation

## 3.1 Input documents

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Specification of Service Discovery
AUTOSAR_CP_SWS_ServiceDiscovery

[3] General Specification of Transformers
AUTOSAR_CP_SWS_TransformerGeneral

[4] Specification of Socket Adaptor
AUTOSAR_CP_SWS_SocketAdaptor

[5] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[6] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[7] Requirements on Transformer
AUTOSAR_CP_RS_Transformer

[8] System Template
AUTOSAR_CP_TPS_SystemTemplate

[9] Specification of Platform Types for Classic Platform
AUTOSAR_CP_SWS_PlatformTypes

[10] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

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

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

[13] SOME/IP Protocol Specification
AUTOSAR_FO_PRS_SOMEIPProtocol

[14] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

## 3.2 Related standards and norms

Not applicable.

## 3.3 Related specification

AUTOSAR provides a General Specification on Transformers [3, SWS Transformer General], which is also valid for SOME/IP Transformer.

Thus, the specification SWS Transformer General shall be considered as additional and required specification for SOME/IP Transformer.

# 4 Constraints and assumptions

## 4.1 Limitations

For the SOME/IP Transformer all general transformer limitations (see [3, SWS Transformer General]) apply.

The SOME/IP transformer doesn't implement the whole SOME/IP protocol:

- a part is implemented by [2, SWS Service Discovery]
- a part is implemented by [4, SWS Socket Adaptor]
- a part is currently not implemented in AUTOSAR. This is documented in Appendix D
- The processing order of header fields in AUTOSAR CP deviates from the processing order defined in [PRS_SOMEIP_00195] (also Figure 4.21: Message Validation and Error Handling in SOME/IP). This deviation is caused by the layered architecture of AUTOSAR CP.

**[CP_SWS_SomeIpXf_CONSTR_00001] Value of length field** ⌈In accordance with [SWS_SomeIpXf_00245], 2^(8*sizeof(data type of length field)) shall be larger than the number of elements given by the size indicator multiplied by the size in bytes of each element (i.e., 1 for UTF-8 and 2 for UTF-16) and increased by the size in bytes required by the BOM.⌋

**[CP_SWS_SomeIpXf_CONSTR_00002] Serialization based on the network representation** ⌈Serialization based on the network representation according to [TPS_SYST_02136] is currently not supported in combination with structured datatypes and arguments with identifier and optional members, strings, dynamic length arrays / variable size arrays, and unions / variants and shall therefore not be used in those combinations.⌋

**Note:**
Optional members according to section 7.1.4.3, Strings according to section 7.1.4.4.1 and 7.1.4.4.2, Dynamic Length Arrays / Variable Size Arrays according to section 7.1.4.7 and Unions / Variants according to section 7.1.4.9.

## 4.2 Applicability to car domains

The SOME/IP Transformer can be used for all domain applications when SOME/IP Sender/Receiver or Client/Server communication is used.

# 5 Dependencies to other modules

The AUTOSAR RTE [5, SWS RTE] has to exist to execute the transformer.

## 5.1 File structure

### 5.1.1 Code file structure

The source code file structure is defined in the [3, SWS Transformer General].

# 6 Requirements Tracing

The following table references the features specified in [6] and [7] and links to the fulfillments of these.

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00004]** | All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00005]** | Modules of the $\mu$C Abstraction Layer (MCAL) may not have hard coded horizontal interfaces | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00159]** | All modules of the AUTOSAR Basic Software shall support a tool based configuration | [SWS_SomeIpXf_00185] |
| **[SRS_BSW_00161]** | The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00162]** | The AUTOSAR Basic Software shall provide a hardware abstraction layer | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00168]** | SW components shall be tested by a function defined in a common API in the Basis-SW | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00170]** | The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands | [SWS_SomeIpXf_00115] [SWS_SomeIpXf_91003] |
| **[SRS_BSW_00310]** | API naming convention | [SWS_SomeIpXf_00115] |
| **[SRS_BSW_00331]** | All Basic Software Modules shall strictly separate error and status information | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00336]** | Basic SW module shall be able to shutdown | [SWS_SomeIpXf_00182] |
| **[SRS_BSW_00337]** | Classification of development errors | [SWS_SomeIPxf_00184] |
| **[SRS_BSW_00345]** | BSW Modules shall support pre-compile configuration | [SWS_SomeIpXf_00182] |
| **[SRS_BSW_00350]** | All AUTOSAR Basic Software Modules shall allow the enabling/ disabling of detection and reporting of development errors. | [SWS_SomeIpXf_00181] [SWS_SomeIpXf_91003] |
| **[SRS_BSW_00351]** | Encapsulation of compiler specific methods to map objects | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00357]** | For success/failure of an API call a standard return type shall be defined | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] |
| **[SRS_BSW_00358]** | The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00369]** | All AUTOSAR Basic Software Modules shall not return specific development error codes via the API | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] |
| **[SRS_BSW_00375]** | Basic Software Modules shall report wake-up reasons | [SWS_SomeIpXf_00100] |

$\bigtriangledown$

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00383]** | The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description | [SWS_SomeIpXf_00144] |
| **[SRS_BSW_00384]** | The Basic Software Module specifications shall specify at least in the description which other modules they require | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00385]** | List possible error notifications | [SWS_SomeIpXf_00115] |
| **[SRS_BSW_00386]** | The BSW shall specify the configuration and conditions for detecting an error | [SWS_SomeIpXf_00115] |
| **[SRS_BSW_00388]** | Containers shall be used to group configuration parameters that are defined for the same object | [SWS_SomeIpXf_00183] |
| **[SRS_BSW_00389]** | Containers shall have names | [SWS_SomeIpXf_00183] |
| **[SRS_BSW_00390]** | Parameter content shall be unique within the module | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00392]** | Parameters shall have a type | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] |
| **[SRS_BSW_00393]** | Parameters shall have a range | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00395]** | The Basic Software Module specifications shall list all configuration parameter dependencies | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00396]** | The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00398]** | The link-time configuration is achieved on object code basis in the stage after compiling and before linking | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00399]** | Parameter-sets shall be located in a separate segment and shall be loaded after the code | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00401]** | Documentation of multiple instances of configuration parameters shall be available | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00403]** | The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00404]** | BSW Modules shall support post-build configuration | [SWS_SomeIpXf_00183] |
| **[SRS_BSW_00411]** | AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of API(s) | [SWS_SomeIpXf_00181] [SWS_SomeIpXf_00182] |
| **[SRS_BSW_00413]** | An index-based accessing of the instances of BSW modules shall be done | [SWS_SomeIpXf_00181] |

▽

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00416]** | The sequence of modules to be initialized shall be configurable | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00417]** | Software which is not part of the SW-C shall report error events only after the Dem is fully operational. | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] |
| **[SRS_BSW_00419]** | If a pre-compile time configuration parameter is implemented as `const` it should be placed into a separate c-file | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00422]** | Pre-de-bouncing of error status information is done within the Dem | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] |
| **[SRS_BSW_00425]** | The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects | [SWS_SomeIpXf_00172] |
| **[SRS_BSW_00432]** | Modules should have separate main processing functions for read/receive and write/transmit data path | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00441]** | Naming convention for type, macro and function | [SWS_SomeIpXf_00183] |
| **[SRS_BSW_00448]** | Module SWS shall not contain requirements from other modules | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00452]** | Classification of runtime errors | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00453]** | BSW Modules shall be harmonized | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00454]** | An alternative interface without a parameter of category DATA_ REFERENCE shall be available. | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00456]** | A Header file shall be defined in order to harmonize BSW Modules | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00458]** | Classification of production errors | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00459]** | It shall be possible to concurrently execute a service offered by a BSW module in different partitions | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00461]** | Modules called by generic modules shall satisfy all interfaces requested by the generic module | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00462]** | All Standardized Autosar Interfaces shall have unique requirement Id / number | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00469]** | Fault detection and healing of production errors | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00470]** | Execution frequency of production error detection | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00471]** | Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00472]** | Avoid detection of two production errors with the same root cause. | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00478]** | Timing limits of main functions | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00479]** | Interfaces for handling request from external devices | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00480]** | Null pointer errors shall follow a naming rule | [SWS_SomeIpXf_00181] |

▽

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00481]** | Invalid configuration set selection errors shall follow a naming rule | [SWS_SomeIpXf_00111] |
| **[SRS_BSW_00483]** | BSW Modules shall handle buffer alignments internally | [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00484]** | Input parameters of scalar and enum types shall be passed as a value. | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00485]** | Input parameters of structure type shall be passed as a reference to a constant structure | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00486]** | Input parameters of array type shall be passed as a reference to the constant array base type | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00181] |
| **[SRS_BSW_00490]** | List possible security events | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00491]** | Specification of trigger conditions and context data | [SWS_SomeIpXf_00100] |
| **[SRS_BSW_00494]** | ServiceInterface argument with a pointer datatype | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00181] |
| **[SRS_Xfrm_00001]** | A transformer shall work on data given by the Rte | [SWS_SomeIpXf_00264] [SWS_SomeIpXf_00265] [SWS_SomeIpXf_00266] |
| **[SRS_Xfrm_00002]** | A transformer shall provide fixed interfaces | [SWS_SomeIpXf_00206] [SWS_SomeIpXf_00207] [SWS_SomeIpXf_00208] [SWS_SomeIpXf_00209] [SWS_SomeIpXf_00210] [SWS_SomeIpXf_00211] [SWS_SomeIpXf_00214] [SWS_SomeIpXf_00296] [SWS_SomeIpXf_00297] [SWS_SomeIpXf_00298] [SWS_SomeIpXf_00299] [SWS_SomeIpXf_00301] [SWS_SomeIpXf_00302] [SWS_SomeIpXf_00303] [SWS_SomeIpXf_00304] [SWS_SomeIpXf_00305] [SWS_SomeIpXf_91001] [SWS_SomeIpXf_91002] |
| **[SRS_Xfrm_00004]** | A transformer shall support error handling | [SWS_SomeIpXf_00264] [SWS_SomeIpXf_00265] [SWS_SomeIpXf_00266] |
| **[SRS_Xfrm_00005]** | A transformer shall be able to deal with more data than expected | [SWS_SomeIpXf_00152] |
| **[SRS_Xfrm_00006]** | A Transformer shall support concurrent execution | [SWS_SomeIpXf_00181] |
| **[SRS_Xfrm_00007]** | A deserializer transformer shall support extraction of data | [SWS_SomeIpXf_00144] |
| **[SRS_Xfrm_00008]** | A transformer shall specify its output format | [SWS_SomeIpXf_00015] [SWS_SomeIpXf_00024] [SWS_SomeIpXf_00025] [SWS_SomeIpXf_00026] [SWS_SomeIpXf_00029] [SWS_SomeIpXf_00030] [SWS_SomeIpXf_00031] [SWS_SomeIpXf_00033] [SWS_SomeIpXf_00152] [SWS_SomeIpXf_00154] [SWS_SomeIpXf_00155] [SWS_SomeIpXf_00156] [SWS_SomeIpXf_00160] [SWS_SomeIpXf_00161] [SWS_SomeIpXf_00163] [SWS_SomeIpXf_00164] [SWS_SomeIpXf_00165] [SWS_SomeIpXf_00166] [SWS_SomeIpXf_00168] [SWS_SomeIpXf_00172] [SWS_SomeIpXf_00212] [SWS_SomeIpXf_00213] [SWS_SomeIpXf_00234] [SWS_SomeIpXf_00235] [SWS_SomeIpXf_00236] [SWS_SomeIpXf_00237] [SWS_SomeIpXf_00238] [SWS_SomeIpXf_00326] |
| **[SRS_Xfrm_00009]** | A fixed set of transformer classes shall exist | [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00144] |
| **[SRS_Xfrm_00010]** | Each transformer class shall provide a fixed set of abstract errors | [SWS_SomeIpXf_00181] |

▽

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_Xfrm_00011]** | A transformer shall belong to a specific transformer class | [SWS_SomeIpXf_00181] |
| **[SRS_Xfrm_00101]** | The SOME/IP Transformer shall define the serialization of atomic and structured data elements into linear arrays | [SWS_SomeIpXf_00016] [SWS_SomeIpXf_00017] [SWS_SomeIpXf_00034] [SWS_SomeIpXf_00036] [SWS_SomeIpXf_00037] [SWS_SomeIpXf_00042] [SWS_SomeIpXf_00053] [SWS_SomeIpXf_00054] [SWS_SomeIpXf_00055] [SWS_SomeIpXf_00056] [SWS_SomeIpXf_00057] [SWS_SomeIpXf_00058] [SWS_SomeIpXf_00059] [SWS_SomeIpXf_00060] [SWS_SomeIpXf_00069] [SWS_SomeIpXf_00070] [SWS_SomeIpXf_00072] [SWS_SomeIpXf_00076] [SWS_SomeIpXf_00088] [SWS_SomeIpXf_00098] [SWS_SomeIpXf_00099] [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00140] [SWS_SomeIpXf_00141] [SWS_SomeIpXf_00143] [SWS_SomeIpXf_00148] [SWS_SomeIpXf_00151] [SWS_SomeIpXf_00169] [SWS_SomeIpXf_00215] [SWS_SomeIpXf_00216] [SWS_SomeIpXf_00217] [SWS_SomeIpXf_00218] [SWS_SomeIpXf_00219] [SWS_SomeIpXf_00220] [SWS_SomeIpXf_00221] [SWS_SomeIpXf_00222] [SWS_SomeIpXf_00223] [SWS_SomeIpXf_00224] [SWS_SomeIpXf_00225] [SWS_SomeIpXf_00226] [SWS_SomeIpXf_00227] [SWS_SomeIpXf_00229] [SWS_SomeIpXf_00230] [SWS_SomeIpXf_00231] [SWS_SomeIpXf_00232] [SWS_SomeIpXf_00233] [SWS_SomeIpXf_00234] [SWS_SomeIpXf_00235] [SWS_SomeIpXf_00236] [SWS_SomeIpXf_00237] [SWS_SomeIpXf_00238] [SWS_SomeIpXf_00240] [SWS_SomeIpXf_00241] [SWS_SomeIpXf_00242] [SWS_SomeIpXf_00243] [SWS_SomeIpXf_00244] [SWS_SomeIpXf_00245] [SWS_SomeIpXf_00246] [SWS_SomeIpXf_00247] [SWS_SomeIpXf_00248] [SWS_SomeIpXf_00249] [SWS_SomeIpXf_00250] [SWS_SomeIpXf_00251] [SWS_SomeIpXf_00252] [SWS_SomeIpXf_00253] [SWS_SomeIpXf_00254] [SWS_SomeIpXf_00256] [SWS_SomeIpXf_00257] [SWS_SomeIpXf_00258] [SWS_SomeIpXf_00259] [SWS_SomeIpXf_00260] [SWS_SomeIpXf_00262] [SWS_SomeIpXf_00263] [SWS_SomeIpXf_00300] [SWS_SomeIpXf_00306] [SWS_SomeIpXf_00307] [SWS_SomeIpXf_00311] [SWS_SomeIpXf_00316] [SWS_SomeIpXf_00317] [SWS_SomeIpXf_00319] [SWS_SomeIpXf_00320] [SWS_SomeIpXf_00321] [SWS_SomeIpXf_00322] [SWS_SomeIpXf_00323] [SWS_SomeIpXf_00324] [SWS_SomeIpXf_00325] [SWS_SomeIpXf_00326] |
| **[SRS_Xfrm_00102]** | The SOME/IP Transformer shall define a protocol for inter-ECU Client/Server communication | [SWS_SomeIpXf_00106] [SWS_SomeIpXf_00107] [SWS_SomeIpXf_00108] [SWS_SomeIpXf_00111] [SWS_SomeIpXf_00112] [SWS_SomeIpXf_00113] [SWS_SomeIpXf_00115] [SWS_SomeIpXf_00120] [SWS_SomeIpXf_00121] [SWS_SomeIpXf_00139] [SWS_SomeIpXf_00170] [SWS_SomeIpXf_00176] [SWS_SomeIpXf_00200] [SWS_SomeIpXf_00201] [SWS_SomeIpXf_00202] [SWS_SomeIpXf_00204] [SWS_SomeIpXf_00205] [SWS_SomeIpXf_00228] [SWS_SomeIpXf_00310] [SWS_SomeIpXf_00312] [SWS_SomeIpXf_00313] |
| **[SRS_Xfrm_00103]** | The SOME/IP Transformer shall support exception notification of applications | [SWS_SomeIpXf_00111] [SWS_SomeIpXf_00310] |

▽

△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_Xfrm_00105]** | The SOME/IP Transformer shall support autonomous error reactions on the server side for client/server communication | [SWS_SomeIpXf_00203] |
| **[SRS_Xfrm_00106]** | The SOME/IP Transformer shall support serialization of extensible data structs and methods | [SWS_SomeIpXf_00142] [SWS_SomeIpXf_00145] [SWS_SomeIpXf_00146] [SWS_SomeIpXf_00147] [SWS_SomeIpXf_00149] [SWS_SomeIpXf_00150] [SWS_SomeIpXf_00267] [SWS_SomeIpXf_00268] [SWS_SomeIpXf_00269] [SWS_SomeIpXf_00270] [SWS_SomeIpXf_00271] [SWS_SomeIpXf_00272] [SWS_SomeIpXf_00273] [SWS_SomeIpXf_00274] [SWS_SomeIpXf_00275] [SWS_SomeIpXf_00276] [SWS_SomeIpXf_00277] [SWS_SomeIpXf_00278] [SWS_SomeIpXf_00279] [SWS_SomeIpXf_00280] [SWS_SomeIpXf_00281] [SWS_SomeIpXf_00282] [SWS_SomeIpXf_00283] [SWS_SomeIpXf_00284] [SWS_SomeIpXf_00285] [SWS_SomeIpXf_00286] [SWS_SomeIpXf_00287] [SWS_SomeIpXf_00288] [SWS_SomeIpXf_00289] [SWS_SomeIpXf_00290] [SWS_SomeIpXf_00291] [SWS_SomeIpXf_00292] [SWS_SomeIpXf_00293] [SWS_SomeIpXf_00294] [SWS_SomeIpXf_00295] [SWS_SomeIpXf_00314] [SWS_SomeIpXf_00315] [SWS_SomeIpXf_00318] |
| **[SRS_Xfrm_00201]** | The COM Based Transformer shall define the serialization of atomic and structured data elements into linear arrays based on a fixed data mapping | [SWS_SomeIpXf_00036] [SWS_SomeIpXf_00181] |

**Table 6.1: Requirements Tracing**

# 7 Functional specification

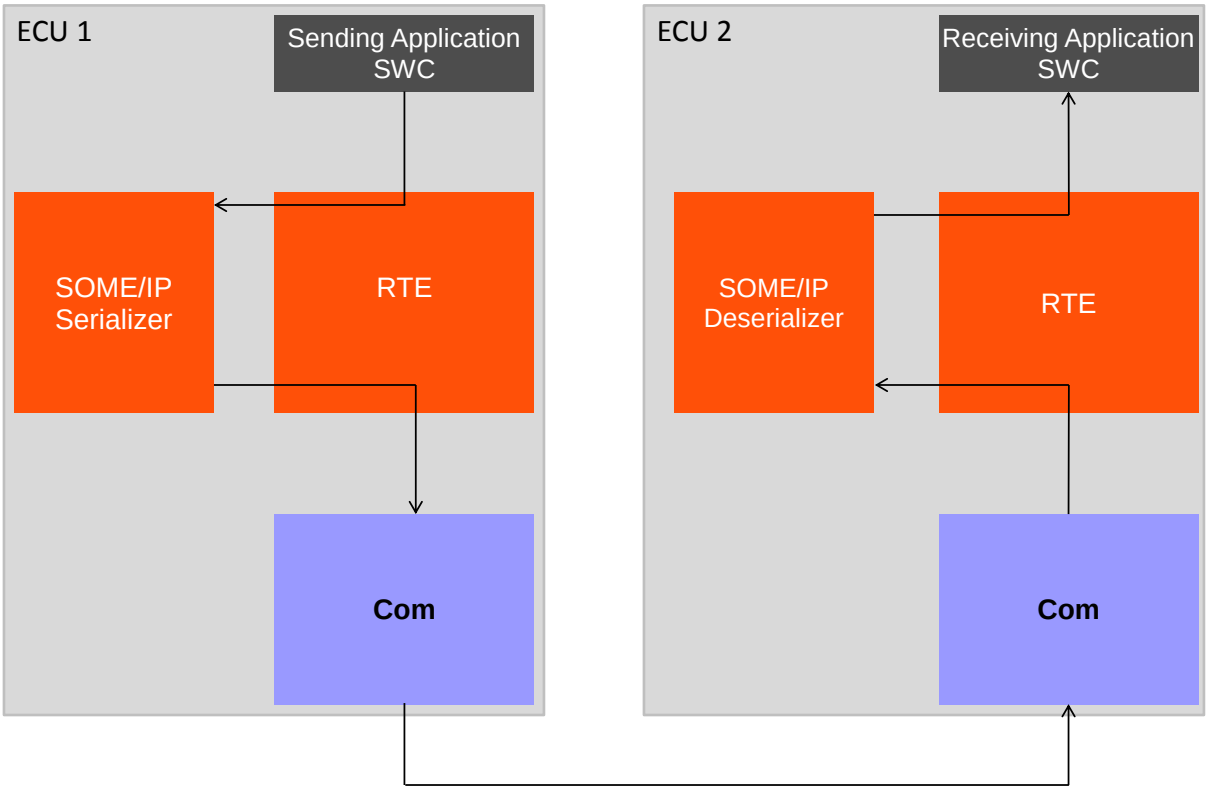


**Figure 7.1: Overview of SOME/IP Transformer**

When a SWC initiates an inter-ECU communication which is configured to be transformed, the SWC hands the data over to the RTE. The RTE executes the configured transformer chain which contains the SOME/IP Transformer (A transformer chain may contain also other transformers but this is omitted in this overview for simplicity).

The SOME/IP Transformer on the sender side serializes the data of the SWC and brings them into an linear form. The serialized data are sent via the communication stack over the bus to the receiver(s). The RTE of the receiver executes the transformer chain in the reverse order. The SOME/IP transformer of the receiver deserializes the linear data back into the original data structure. These are handed over to the receiving SWC.

From the SWC's point of view it is totally transparent whether data are transformed or not.

The SOME/IP transformer is a transformer of the class **Serializer**. It serializes structured data into a linear form. Therefore it can only be used as the first transformer on the sending side and the last transformer on the receiving side (in execution order). Furthermore it provides the transformer errors specified for this transformer class and supports only out-of-place buffer handling.

The SOME/IP Transformer has no module specific EcuC because its whole configuration is based on the SOMEIPTransformationDescription and SOMEIPTransformationISignalProps.

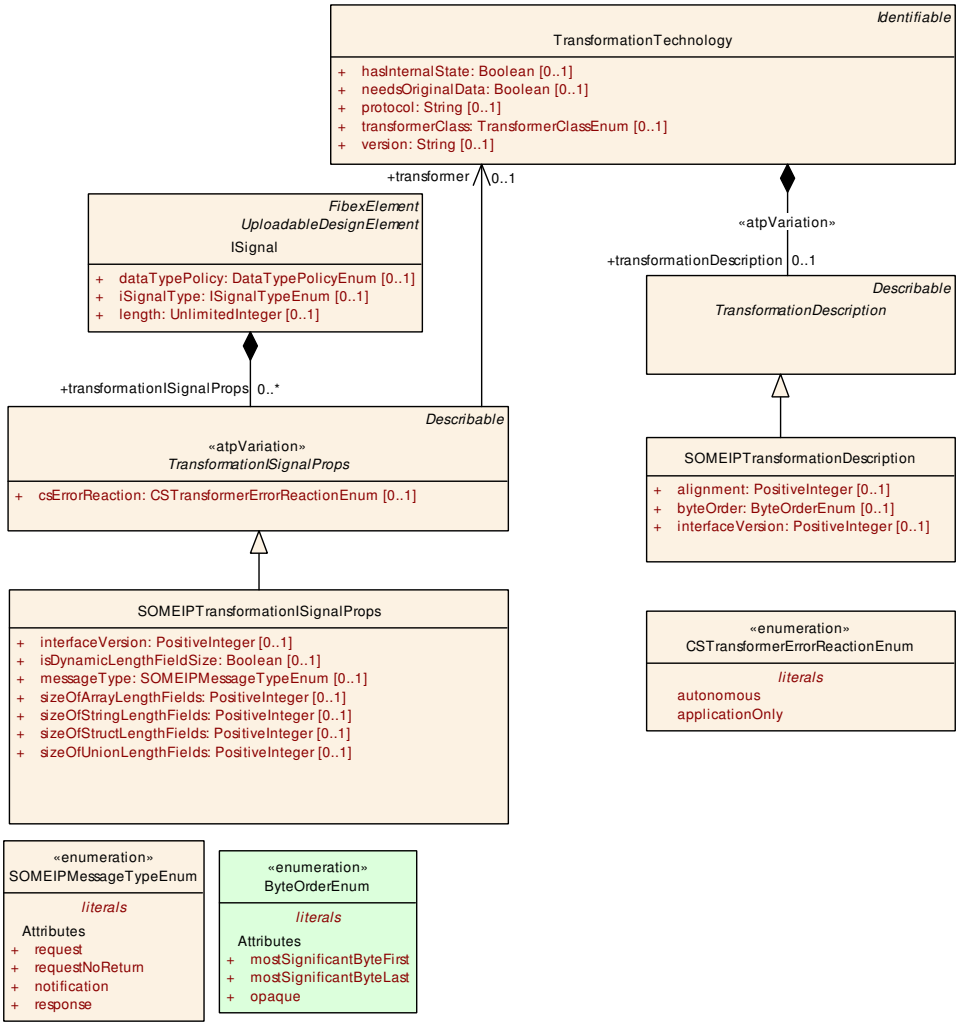


**Figure 7.2: SOME/IP specific configuration**

| ***Class*** | **SOMEIPTransformationDescription** | | | |
|---|---|---|---|---|
| ***Note*** | The SOMEIPTransformationDescription is used to specify SOME/IP transformer specific attributes. | | | |
| ***Base*** | *ARObject*, *Describable*, *TransformationDescription* | | | |
| ***Aggregated by*** | TransformationTechnology.transformationDescription | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| alignment | PositiveInteger | 0..1 | attr | Defines the padding for alignment purposes that will be added by the SOME/IP transformer after the serialized data of the variable data length data element. The alignment shall be specified in Bits. |
| byteOrder | ByteOrderEnum | 0..1 | attr | Defines which byte order shall be serialized by the SOME/IP transformer |
| interfaceVersion | PositiveInteger | 0..1 | attr | The interface version the SOME/IP transformer shall use. |

**Table 7.1: SOMEIPTransformationDescription**

| *Class* | «atpVariation» **SOMEIPTransformationISignalProps** | | | |
|---|---|---|---|---|
| ***Note*** | The class SOMEIPTransformationISignalProps specifies ISignal specific configuration properties for SOME/IP transformer attributes. | | | |
| ***Base*** | *ARObject*, *Describable*, *TransformationISignalProps* | | | |
| ***Aggregated by*** | ISignal.transformationISignalProps, ISignalGroup.transformationISignalProps | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| interfaceVersion | PositiveInteger | 0..1 | attr | The interface version the SOME/IP transformer shall use. |
| isDynamic LengthFieldSize | Boolean | 0..1 | attr | This attribute shall be used to determine the wire type in the context of using the TLV encoding. |
| messageType | SOMEIPMessageType Enum | 0..1 | attr | The Message Type which shall be placed into the SOME/IP header. |
| sizeOfArray LengthFields | PositiveInteger | 0..1 | attr | The size of all length fields (in Bytes) of fixed-size arrays or dynamic size arrays in the SOME/IP message. This attribute is valid for all available occurrences of fixed-size arrays or dynamic size arrays in the SOME/IP message. |
| sizeOfString LengthFields | PositiveInteger | 0..1 | attr | The size of all length fields (in Bytes) of dynamic or fixed length strings in the SOME/IP message. This attribute is valid for all available occurrences of strings in the SOME/IP message. |
| sizeOfStruct LengthFields | PositiveInteger | 0..1 | attr | The size of all length fields (in Bytes) of structs in the SOME/IP message. This attribute is valid for all available occurrences of structures in the SOME/IP message. For a more fine granular modeling on the level of Data Prototypes the DataPrototypeTransformationProps shall be used. |
| sizeOfUnion LengthFields | PositiveInteger | 0..1 | attr | The size of all length fields (in Bytes) of unions in the SOME/IP message. This attribute is valid for all available occurrences of Unions in the SOME/IP message. For a more fine granular modeling on the level of Data Prototypes the DataPrototypeTransformationProps shall be used. |
| tlvDataId Definition | TlvDataIdDefinitionSet | * | ref | This reference identifies the TlvDataIdDefinitions relevant for the enclosing SOMEIPTransformationISignalProps<br>**Tags:** atp.Splitkey=tlvDataIdDefinition |

**Table 7.2: SOMEIPTransformationISignalProps**

| *Enumeration* | **ByteOrderEnum** |
|---|---|
| ***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.<br>ByteOrder is very important in case of communication between different PUs or ECUs. |
| ***Aggregated by*** | ApSomeipTransformationProps.byteOrder, BaseTypeDirectDefinition.byteOrder, DiagnosticCommonProps.defaultEndianness, ISignalToIPduMapping.packingByteOrder, MultiplexedIPdu.selectorFieldByteOrder, PduToFrameMapping.packingByteOrder, SegmentPosition.segmentByteOrder, SOMEIPTransformationDescription.byteOrder, System.containerIPduHeaderByteOrder |
| ***Literal*** | ***Description*** |
| mostSignificantByteFirst | Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format)<br>**Tags:** atp.EnumerationLiteralIndex=0 |
| mostSignificantByteLast | Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format)<br>**Tags:** atp.EnumerationLiteralIndex=1 |
| opaque | For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details.<br>**Tags:** atp.EnumerationLiteralIndex=2 |

**Table 7.3: ByteOrderEnum**

| *Enumeration* | **SOMEIPMessageTypeEnum** |
|---|---|
| ***Note*** | Depending on the style of the communication different message types shall be set in the header of a SOME/IP message. |
| ***Aggregated by*** | SOMEIPTransformationISignalProps.messageType |
| ***Literal*** | ***Description*** |
| notification | A request of a notification expecting no response.<br>**Tags:** atp.EnumerationLiteralIndex=1 |
| request | A request expecting a response.<br>**Tags:** atp.EnumerationLiteralIndex=2 |
| requestNoReturn | A fire&forget request.<br>**Tags:** atp.EnumerationLiteralIndex=3 |
| response | The response message.<br>**Tags:** atp.EnumerationLiteralIndex=4 |

**Table 7.4: SOMEIPMessageTypeEnum**

**[SWS_SomeIpXf_00151]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The SOME/IP transformer defined in this document shall be used as a transformer if

- the attribute `protocol` of the `TransformationTechnology` is set to `SOMEIP`
- and the attribute `version` of the `TransformationTechnology` is set to `1`
- and the attribute `transformerClass` of the `TransformationTechnology` is set to `serializer`

⌋

## 7.1 Specification of the SOME/IP on-wire format

Serialization describes the way data is represented in protocol data units (PDUs) transported over an automotive in-vehicle network.

### 7.1.1 Message Length Limitations

The usage of TCP allows for larger streams of data to transport SOME/IP header and payload. However, current transport protocols for CAN and FlexRay limit messages to 4095 Bytes. When compatibility to those has to be achieved, SOME/IP messages including the SOME/IP header shall not exceed 4095 Bytes.

### 7.1.2 Endianness

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

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

**[SWS_SomeIpXf_00172]**

*Upstream requirements:* SRS_Xfrm_00008, SRS_BSW_00425

⌈The byte order of the parameters inside the payload shall be defined by `byteOrder` of `SOMEIPTransformationDescription`. If `byteOrder` of `SOMEIPTransformationDescription` is not configured explicitly, the default value defined for `BYTE_ORDER` in [PRS_SOMEIP_00369] shall be used.⌋

### 7.1.3 Message format

**[SWS_SomeIpXf_00152]**

*Upstream requirements:* SRS_Xfrm_00008, SRS_Xfrm_00005

⌈For interoperability reasons the message format layout shall be identical for all implementations of SOME/IP and is described as follows:

1. Message ID (Service ID / Method ID) [32 bit]
2. Length [32 bit]
3. Additional information:
   (a) Protocol Version [8 bit]
   (b) Interface Version [8 bit]
   (c) Message Type [8 bit]
   (d) Return Code [8 bit]
4. Payload [variable size]

The fields are presented in transmission order; i.e. the fields on the top are transmitted first. In the following sections the different message format fields and their usage is being described.⌋

**Note:**
Layout is also shown in Figure 7.3.

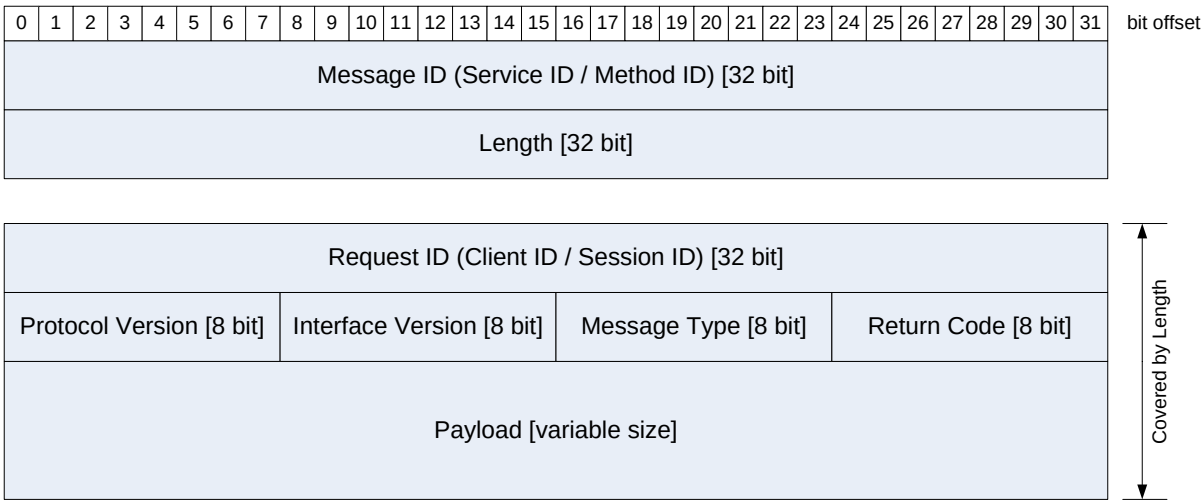


**Figure 7.3: SOME/IP Message Format**

Figure 7.3 shows the **complete** SOME/IP message format. The SOME/IP transformer only implements the lower part (all except Message ID and Length).

**[SWS_SomeIpXf_00015]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The SOME/IP transformer shall implement all fields of the header except Message ID and Length.⌋

Rationale:
Message-ID and Length are not covered since this allows the AUTOSAR Socket Adaptor header mode to work.

These are added by other modules in the AUTOSAR BSW. Nonetheless they are contained in Figure 7.3 to show the whole on-wire-format.

#### 7.1.3.1 Request ID [32 bit]

**[SWS_SomeIpXf_00154]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Request ID field shall be 32 bit long.⌋

The Request ID shall be the unique identifier for the calling client inside the ECU. Its values are chosen by the RTE and handed over to the SOME/IP transformer.

**[SWS_SomeIpXf_00024] Request ID construction by Client ID and Session ID**

*Upstream requirements:* SRS_Xfrm_00008

⌈

| Client ID [16 bits] | Session ID [16 bits] |
|---|---|

⌋

Both are chosen by RTE and handed over to the transformer as `Rte_Cs_TransactionHandleType`.

**[SWS_SomeIpXf_00025]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The `clientId` inside the `Rte_Cs_TransactionHandleType` handed over from RTE shall be used for the value of the Client ID.⌋

**[SWS_SomeIpXf_00026]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The `sequenceCounter` inside the `Rte_Cs_TransactionHandleType` handed over from RTE shall be used for the value of the Session ID.⌋

For details of `Rte_Cs_TransactionHandleType` see [SWS_Rte_08732].

The Request ID allows a client to differentiate multiple calls to the same method. Therefore, the Request ID has to be unique for a single client and server combination only. When generating a response message, the server has to copy the Request ID from the request to the response message. This allows the client to map a response to the issued request even with more than one request outstanding.

Request IDs may be reused as soon as the response arrived or is not expected to arrive anymore (timeout).

#### 7.1.3.2 Protocol Version [8 bit]

**[SWS_SomeIpXf_00155]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Protocol Version field shall be 8 bit long.⌋

**[SWS_SomeIpXf_00156]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Protocol Version field shall contain the SOME/IP protocol version.⌋

**[SWS_SomeIpXf_00029]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Protocol Version shall be set to 0x01.⌋

#### 7.1.3.3 Interface Version [8 bit]

**[SWS_SomeIpXf_00030]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Interface Version field shall be 8 bit long.⌋

**[SWS_SomeIpXf_00160]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Interface Version field shall contain the Version of the Service Interface.⌋

Rationale: This is required to catch mismatches in Service definitions and allows debugging tools to identify the Service Interface used, if version is used.

**Note:**
The Version of the corresponding Service Discovery service has to match the version of the Service Interface, i.e. `SdServerServiceMajorVersion` and/or `SdClientServiceMajorVersion` has to match the used `SOMEIPTransformationDescription.interfaceVersion` and/or `SOMEIPTransformationISignalProps.interfaceVersion` (see [TPS_SYST_02377]).

#### 7.1.3.4 Message Type [8 bit]

**[SWS_SomeIpXf_00161]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Message Type field shall be 8 bit long.⌋

The Message Type field is used to differentiate different types of messages.

**[SWS_SomeIpXf_00031] Message TYPE field values**

*Upstream requirements:* SRS_Xfrm_00008

⌈

| Number | Value | Description |
|---|---|---|
| 0x00 | REQUEST | A request expecting a response (even void) |
| 0x01 | REQUEST_NO_RETURN | A fire&forget request |
| 0x02 | NOTIFICATION | A request of a notification expecting no response |
| 0x80 | RESPONSE | The response message |
| 0x81 | ERROR | The response containing an error |

⌋

Refer to [PRS_SOMEIP_00701] with the note and text below it, that describes the use or the need of each of the above mentioned message types.

#### 7.1.3.5 Return Code [8 bit]

**[SWS_SomeIpXf_00163]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Return Code field shall be 8 bit long.⌋

**[SWS_SomeIpXf_00164]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Return Code field shall be used to signal whether a request has been successfully processed.⌋

For simplification of the header layout, every message transports the field Return Code.

The Return Codes are specified in detail in [SWS_SomeIpXf_00115].

**[SWS_SomeIpXf_00033]**

*Upstream requirements:* SRS_Xfrm_00008

⌈Messages of Type REQUEST, REQUEST_NO_RETURN, and Notification have to set the Return Code to 0x00 (E_OK).⌋

**[SWS_SomeIpXf_00168] Allowed Return Codes for specific message types**

*Upstream requirements:* SRS_Xfrm_00008

⌈

| Message Type | Allowed Return Codes |
|---|---|
| REQUEST | N/A, set to 0x00 (E_OK) |
| REQUEST_NO_RETURN | N/A, set to 0x00 (E_OK) |
| NOTIFICATION | N/A, set to 0x00 (E_OK) |
| RESPONSE | See Return Codes in [SWS_SomeIpXf_00115]. |

⌋

#### 7.1.3.6 Payload [variable size]

**[SWS_SomeIpXf_00165]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Payload field shall have variable size.⌋

**[SWS_SomeIpXf_00166]**

*Upstream requirements:* SRS_Xfrm_00008

⌈The Payload field shall contain the transported data.⌋

The serialization of the data will be specified in the following section.

### 7.1.4 Serialization of Parameters and Data Structures

**[SWS_SomeIpXf_00034]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The serialization shall be based on the `SenderReceiverInterface` or `ClientServerInterface` of the data.⌋

**[SWS_SomeIpXf_00259]**

*Upstream requirements:* SRS_Xfrm_00101

⌈After the serialized data of a variable data length `DataPrototype` a padding for alignment purposes shall be added for the configured alignment (see [SWS_SomeIpXf_00260] and [SWS_SomeIpXf_00262]) if the variable data length `DataPrototype` is not the last element in the serialized data stream. This requirement does not apply for the serialization of extensible structs and methods.⌋

**Note:**
See also chapter 7.1.4.3.

**[SWS_SomeIpXf_00260]**

*Upstream requirements:* SRS_Xfrm_00101

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

**Note:**
See also chapter 7.1.4.3.

**[SWS_SomeIpXf_00262]**

*Upstream requirements:* SRS_Xfrm_00101

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

**Note:**
See also chapter 7.1.4.3.

**[SWS_SomeIpXf_00316] Default for alignment**

*Upstream requirements:* SRS_Xfrm_00101

⌈If `SOMEIPTransformationProps.alignment` and `SOMEIPTransformationDescription.alignment` are not configured explicitly, the default value for `ALIGNMENT` defined in [PRS_SOMEIP_00613], converted to bits shall be used.⌋

**[SWS_SomeIpXf_00263]**

*Upstream requirements:* SRS_Xfrm_00101

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

**Note:**
If the following data element shall be aligned, a padding element of according size needs to be explicitly inserted into the `ImplementationDataType` (in case of serialization based on `ImplementationDataType`s according to [SWS_SomeIpXf_00307]) or into the `AutosarDataType` (in case of serialization based on NetworkRepresentation according to [SWS_SomeIpXf_00306]).

**[SWS_SomeIpXf_00037]**

*Upstream requirements:* SRS_Xfrm_00101

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

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

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

In the following the serialization of different parameters is specified.

**[SWS_SomeIpXf_00306] Serialization based on NetworkRepresentation**

*Upstream requirements:* SRS_Xfrm_00101

⌈If a `networkRepresentationProps` is defined according to [TPS_SYST_02136] on the `ISignal`, then the SOME/IP serialization shall be based on the `networkRepresentationProps`.⌋

**Note:**
For details refer to chapter Network Representation in [8].

**[SWS_SomeIpXf_00307] Serialization based on ImplementationDataTypes**

*Upstream requirements:* SRS_Xfrm_00101

⌈If no `networkRepresentationProps` is defined on the `ISignal`, then (according to [TPS_SYST_02137]) the SOME/IP serialization shall be based on the `ImplementationDataType`s.⌋

#### 7.1.4.1 Basic Datatypes

**[SWS_SomeIpXf_00036] Supported `SwBaseType`s for serialization**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00201

⌈

| 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) |

⌋

Note: The `SwBaseType`s are defined in [9] and according to [TPS_STDT_00067] placed in the package `/AUTOSAR_Platform/BaseTypes` (e.g., `/AUTOSAR_Platform/BaseTypes/uint32`).

The Byte Order is specified common for all parameters by `byteOrder` of `SOMEIPTransformationDescription`. See chapter 7.1.2.

#### 7.1.4.2 Structured Datatypes (structs)

**[SWS_SomeIpXf_00042]**

*Upstream requirements:* SRS_Xfrm_00101

⌈A struct shall be serialized in order of depth-first traversal.⌋

The transformer 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 implementation 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 implementation but only in the tool chain used to generate the implementation.

Messages of legacy busses like CAN and FlexRay are usually not aligned. Warnings can be turned off or be ignored in such cases.

The SOME/IP transformer 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_SomeIpXf_00216]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfStructLengthFields` of `SOMEIPTransformationISignalProps` is set to a value greater 0, a length field shall be inserted in front of every serialized struct.⌋

**Note:**
[SWS_SomeIpXf_00216] also applies to nested structs which means that additionally every nested struct has its own length field. Furthermore, in an array of structs where all structs have the same length, the length field is inserted in front of every struct inside the array.

**[SWS_SomeIpXf_00252]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfStructLengthField` of `SOMEIPTransformationProps` is set to a value greater 0, a length field shall be inserted in front of the serialized struct for which the `SOMEIPTransformationProps` is defined. (See [TPS_SYST_02121])⌋

**Note:**
[SWS_SomeIpXf_00252] applies if the length fields of the struct and all nested structs contained within the root struct are configured to different values for the lengths of the length fields via `SOMEIPTransformationProps`.

**[SWS_SomeIpXf_00217]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The data type of the length field of the struct and all nested structs within the struct shall be the same and shall be determined by the value of `SOMEIPTransformationISignalProps.sizeOfStructLengthFields` of the serialized `ISignal`:

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

⌋

**[SWS_SomeIpXf_00253]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If `SOMEIPTransformationProps.sizeOfStructLengthField` is present for a struct the data type for the length field of the struct shall be determined by the value of `SOMEIPTransformationProps.sizeOfStructLengthField`:

- *uint8* if `sizeOfStructLengthField` equals 1
- *uint16* if `sizeOfStructLengthField` equals 2
- *uint32* if `sizeOfStructLengthField` equals 4
- Otherwise [SWS_SomeIpXf_00217] applies.

⌋

**[SWS_SomeIpXf_00317] Default for sizeOfStructLengthFields**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfStructLengthFields` of `SOMEIPTransformationISignal-Props` is not configured explicitly, the default value for `SIZE_OF_STRUCT_LENGTH_FIELD` defined in [PRS_SOMEIP_00079] shall be used.⌋

**[SWS_SomeIpXf_00218]**

*Upstream requirements:* SRS_Xfrm_00101

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

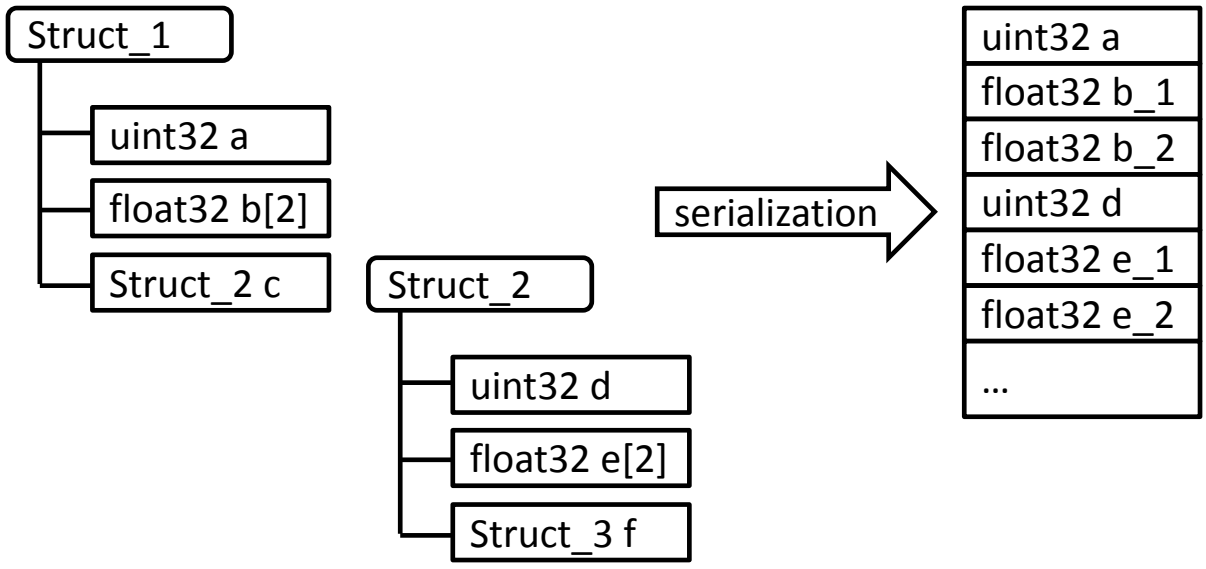


**Figure 7.4: Serialization of Structs without Length Fields (Example)**

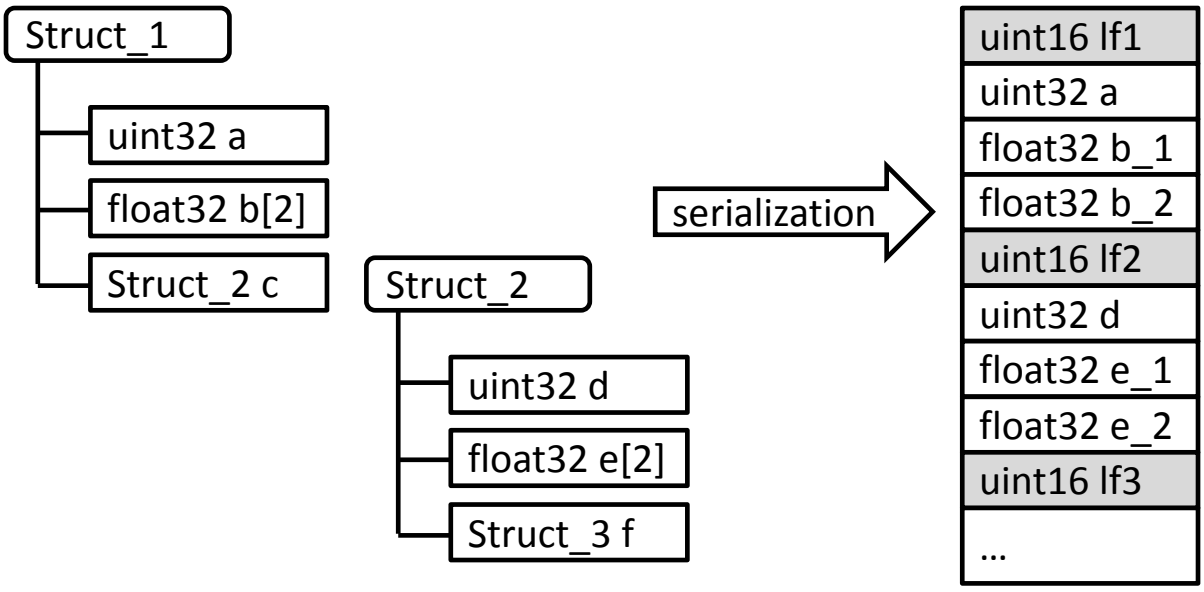


**Figure 7.5: Serialization of Structs with Length Fields (Example)**

#### 7.1.4.3 Structured Datatypes and Arguments with Identifier and optional Members

Please note that the content of this chapter has draft character

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

Structs are modeled in the Software Component Template using an `ImplementationDataType` of category `STRUCTURE` and members are represented by `ImplementationDataTypeElement`s. Method arguments are represented by ArgumentDataPrototypes. Refer to [10] for more details.

The assignment of Data IDs is modeled in the System Template in the context of `SOMEIPTransformationISignalProps`. Refer to [8] for more details.

Moreover, the usage of Data IDs allows describing structs with optional members. To serialize data with optional members, the transformer has to know which optional members are available or not. This is stored in a bitfield which is contained inside the ImplementationDataType. This availabilityBitfield is realized as array of uint8.

Whether an optional member is actually present in the struct or not, must be determined during runtime.

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

**[SWS_SomeIpXf_00267]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The length of a tag shall be two bytes.⌋

**[SWS_SomeIpXf_00268]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The tag shall consist of

- reserved (Bit 7 of the first byte)
- wire type (Bit 6-4 of the first byte)
- Data ID (Bit 3-0 of the first byte and bit 7-0 of the second byte)

Bit 7 is the highest significant bit of a byte, bit 0 is the lowest significant bit of a byte.⌋

**Note:**
Refer to Figure 7.6 for the layout of the tag.

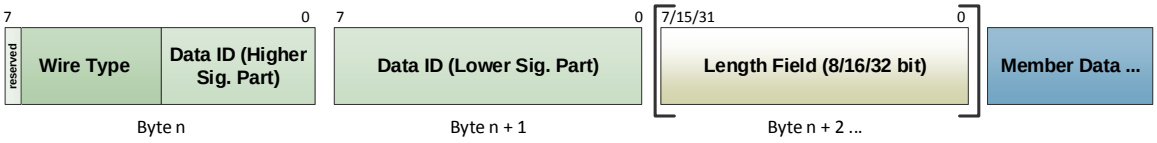


**Figure 7.6: SOME/IP Struct Tag Layout**

**[SWS_SomeIpXf_00269]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The lower significant part of the Data ID of the member shall be encoded in bits 7-0 of the second byte of the tag. The higher significant part of the Data ID of the member shall be encoded in bits 3-0 of the first byte.⌋

Example: The Data ID of the member is 1266 (dec). Then bits 3-0 of the first byte are set to 0x4. The second byte is set to 0xF2.

**[SWS_SomeIpXf_00270] Wire type values determening types of data**

*Upstream requirements:* SRS_Xfrm_00106

⌈

| Wire Type | Value |
|---|---|
| 0 | 8 Bit Data Base data type |
| 1 | 16 Bit Data Base data type |
| 2 | 32 Bit Data Base data type |
| 3 | 64 Bit Data Base data type |
| 4 | Complex Data Type: Array, Struct, String, Union with length field of static size (configured in data definition) |
| 5 | Complex Data Type: Array, Struct, String, Union with length field size 1 byte (ignore static definition) |

| | |
|---|---|
| 6 | Complex Data Type: Array, Struct, String, Union with length field size 2 byte (ignore static definition) |
| 7 | Complex Data Type: Array, Struct, String, Union with length field size 4 byte (ignore static definition) |

⌋

**Note:**
Wire type 4 ensures the compatibility with the current approach where the size of length fields is statically configured. This approach has the drawback that changing the size of the length field during evolution of interfaces is always incompatible. Thus, wire types 5, 6 and 7 allow to encode the size of the used length field in the transferred byte stream.

**[SWS_SomeIpXf_00271]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If `SOMEIPTransformationISignalProps.isDynamicLengthFieldSize` is set to false, the transformer shall use wire type 4 for serializing complex types and shall use the fixed size length fields. The size of the length fields is defined in `SOMEIPTransformationISignalProps.sizeOfArrayLengthFields`, `sizeOfStructLengthFields` and `sizeOfUnionLengthFields`.⌋

**[SWS_SomeIpXf_00272]**

*Upstream requirements:* SRS_Xfrm_00106

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

**[SWS_SomeIpXf_00318] Default for isDynamicLengthFieldSize**

*Upstream requirements:* SRS_Xfrm_00106

⌈If`SOMEIPTransformationISignalProps.isDynamicLengthFieldSize` is not configured explicitly, the default value for `IS_DYNAMIC_LENGTH_FIELD_SIZE` defined in [PRS_SOMEIP_00003] shall be used.⌋

**[SWS_SomeIpXf_00273]**

*Upstream requirements:* SRS_Xfrm_00106

⌈A deserializer shall always be able to deserialize known members/arguments and skip unknown members/arguments with the wire types 4, 5, 6 and 7 independent of the setting of `SOMEIPTransformationISignalProps.isDynamicLengthFieldSize`⌋

**[SWS_SomeIpXf_00274]**

*Upstream requirements:* SRS_Xfrm_00106

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

**Note:**
regarding existence of Data IDs, refer to [8].

**[SWS_SomeIpXf_00275]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the datatype of the serialized member / argument is a basic datatype (wire types 0-3) and a Data ID is configured, the tag shall be inserted directly in front of the member/argument. No length field shall be inserted into the serialized stream.⌋

**[SWS_SomeIpXf_00276]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the datatype of the serialized member/argument is not a basic datatype (wire type 4-7) and a Data ID is configured, the tag shall be inserted in front of the length field.⌋

**[SWS_SomeIpXf_00277]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the datatype of the serialized member/argument is not a basic datatype and a Data ID is configured, a length field shall always be inserted in front of the member/argument.⌋

Rationale: The length field is required for the de-serialization of known members/arguments and to skip unknown members/arguments during deserialization.

**[SWS_SomeIpXf_00278]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The length field shall always contain the length up to the next tag of the struct, but does not include the tag size and length field size itself.⌋

**[SWS_SomeIpXf_00279]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the member itself is of type struct, there shall be exactly one length field.⌋

**[SWS_SomeIpXf_00280]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the member itself is of type dynamic length string, there shall be exactly one length field.⌋

**[SWS_SomeIpXf_00281] Length Field with TLV**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the member itself is of type fixed length string, there shall be exactly one length field corresponding to dynamic length strings.⌋

**Note:**
When serialized without tag, fixed length strings do not have a length field. For the serialization with tag, a length field is also required for fixed length strings in the same way as for dynamic length strings that correspond to the full fixed-length defined in the data type, including BOM and any padding bytes.

**[SWS_SomeIpXf_00282]**

*Status:* DRAFT
*Upstream requirements:* SRS_Xfrm_00106

⌈If the member itself is of type dynamic length array, there shall be exactly one length field.⌋

**[SWS_SomeIpXf_00283]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the member itself is of type fixed length array, there shall be exactly one length field.⌋

**[SWS_SomeIpXf_00284]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the member itself is of type union, there shall be exactly one length field.⌋

**[SWS_SomeIpXf_00285]**

*Upstream requirements:* SRS_Xfrm_00106

⌈For the serialization of extensible structs and methods the length field shall cover the size of the type field, data and padding bytes if the member itself is of type union.⌋

**Note:**
For the serialization without tags, the length field of unions does not cover the type field (see [SWS_SomeIpXf_00226]). For the serialization with tags, it is required that the complete content of the serialized union is covered by the length field.

**[SWS_SomeIpXf_00286]**

*Upstream requirements:* SRS_Xfrm_00106

⌈A member of a non-extensible (standard) struct which is of type extensible struct, shall be serialized according to the requirements for extensible structs.⌋

**[SWS_SomeIpXf_00287]**

*Upstream requirements:* SRS_Xfrm_00106

⌈A member of an extensible struct which is of type non-extensible (standard) struct, shall be serialized according to the requirements for standard structs.⌋

**[SWS_SomeIpXf_00288]**

*Upstream requirements:* SRS_Xfrm_00106

⌈For the serialization of extensible structs and methods no alignment shall be applied.⌋

Rationale: When alignment greater 8 bits is used, the serializer may add padding bytes after variable length data. The padding bytes are not covered by the length field. If the receiver does not know the Data ID of the member, it also does not know that it is variable length data and that there might be padding bytes.

**[SWS_SomeIpXf_00289]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the attribute `isStructWithOptionalElement` of the `ImplementationDataType` representing the extensible struct is set to true, the transformer shall ignore the first ImplementationDataTypeElement and shall not serialize or deserialize it.⌋

Rationale: the first `ImplementationDataTypeElement` represents the availability bitfield which is not transferred on the wire.

**[SWS_SomeIpXf_00290]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The transformer shall only serialize an optional member of a struct if the corresponding bit in the availability bitfield is set as follows:

```
(availabilityBitfield[(pos/8)] & (1<<(pos mod 8))) != 0
```

⌋

**[SWS_SomeIpXf_00291]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If an optional member is available in the serialized byte stream, the transformer shall set the corresponding bit in the availability bitfield as follows:

```
availabilityBitfield[(pos/8)] = availabilityBitfield[(pos/8)] | (1<<(pos mod 8))
```

⌋

**[SWS_SomeIpXf_00292]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If an optional member is not available in the serialized byte stream, the transformer shall clear the corresponding bit in the availability bitfield as follows:

```
availabilityBitfield[(pos/8)] = availabilityBitfield[(pos/8)] & ~(1<<(pos mod 8))
```

⌋

In the requirements [SWS_SomeIpXf_00288], [SWS_SomeIpXf_00289] and [SWS_SomeIpXf_00290] pos is the position of the optional `ImplementationDataTypeElement` among all optional `ImplementationDataTypeElement`s within the `ImplementationDataType` starting with pos = 0.

**Note:**
Non-optional `ImplementationDataTypeElement`s do not count since they do not need a bit in the availabilityBitfield. So the bit position within the availabilityBitfield is determined by the order of the optional `ImplementationDataTypeElement`s. Examples:

- 1st optional ImplementationDataTypeElement (pos=0):

  ```
  (availabilityBitfield[0] & 0x01) != 0
  ```

- 8th optional ImplementationDataTypeElement (pos=7):

  ```
  (availabilityBitfield[0] & 0x80) != 0
  ```

- 9th optional ImplementationDataTypeElement (pos=8):

  ```
  (availabilityBitfield[1] & 0x01) != 0
  ```

**[SWS_SomeIpXf_00295]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If an optional member is not available in the received serialized byte stream, the transformer shall keep the memory section occupied by this optional element without modification.⌋

**[SWS_SomeIpXf_00293]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the transformer reads an unknown Data ID (i.e. not contained in its data definition), it shall skip the unknown member/argument by using the information of the wire type and length field.⌋

#### 7.1.4.4 Strings

**[SWS_SomeIpXf_00053]**

*Upstream requirements:* SRS_Xfrm_00101

⌈Strings shall be encoded using Unicode and terminated with a

`"\textbackslash0"-character`

for both fixed-length and dynamic-length strings. Unused space shall be filled using "\0".⌋

**[SWS_SomeIpXf_00319] Default for baseTypeEncoding**

*Upstream requirements:* SRS_Xfrm_00101

⌈If `baseTypeDefinition.baseTypeEncoding` is not configured explicitly, then the default value for `STRING_ENCODING` as defined in [PRS_SOMEIP_00372] shall be used.⌋

**[SWS_SomeIpXf_00054]**

*Upstream requirements:* SRS_Xfrm_00101

⌈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 four times the length of characters in UTF-8 plus 1 Byte for the termination with a "\0" or up to four times the length of the characters in UTF-16 plus 2 Bytes for a "\0". UTF-8 character can be up to 4 bytes and an UTF-16 character can be up to 4 bytes.⌋

In the following an example is provided in accordance with [SWS_SomeIpXf_00245] and [SWS_SomeIpXf_00054] :

- Single UTF character encoded in UTF-8: 1..4 bytes
- Single UTF character encoded in UTF-16: 2 or 4 bytes
- UTF String with n chars encoded in UTF-8 = up to 3 byte BOM + n*4 UTF-8 Char + 1 bytes Zero Termination = up to 4 + 4*n bytes
- UTF String with n chars encoded in UTF-16 = up to 2 byte BOM + n*4 UTF-16 Char + 2 bytes Zero Termination = up to 4 + 4*n bytes

**[SWS_SomeIpXf_00055]**

*Upstream requirements:* SRS_Xfrm_00101

⌈UTF-16LE and UTF-16BE strings shall be zero terminated with a

`"\textbackslash0"-character`

. This means they shall end with (at least) two 0x00 Bytes.⌋

**[SWS_SomeIpXf_00056]**

*Upstream requirements:* SRS_Xfrm_00101

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

**[SWS_SomeIpXf_00057]**

*Upstream requirements:* SRS_Xfrm_00101

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

**[SWS_SomeIpXf_00248]**

*Upstream requirements:* SRS_Xfrm_00101

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

**[SWS_SomeIpXf_00058]**

*Upstream requirements:* SRS_Xfrm_00101

⌈All strings shall always start with a Byte Order Mark (BOM). The BOM shall be included in fixed-length-strings as well as dynamic-length strings.⌋

For the specification of BOM, see [11] and [12]. Please note that the BOM is used in the serialized strings to achieve compatibility with Unicode.

For the details of the recognition and serialization of fixed- and dynamic-length strings see chapter 7.1.4.4.1 and chapter 7.1.4.4.2.

**[SWS_SomeIpXf_00059]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The receiving SOME/IP transformer implementation shall check the BOM and handle a missing BOM or a malformed BOM as an error.⌋

**[SWS_SomeIpXf_00060] BOM Insertion for Fixed-Length Strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈The `BOM` shall be added by the `SOME/IP` sending transformer implementation.For fixed-length strings, the size required by the `BOM` shall be included in the fixed-length field. If the `BOM`, actual characters, and terminating `NUL` do not fill the fixed-length size, the remaining bytes shall be padded with `NUL`.⌋

### 7.1.4.4.1 Strings (fixed length)

The length of the string (this includes the "\0") in Bytes is specified in the data type definition.

**[SWS_SomeIpXf_00240] Recognition of UTF-8 Fixed Length Strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈An UTF-8 Fixed Length String shall be detected if an `ApplicationPrimitiveDataType` and an `ImplementationDataType` with the following pattern are used:

- `ApplicationPrimitiveDataType`
  - with `category` equal to `STRING`

  - ApplicationPrimitiveDataType.swDataDefProps.swTextProps.baseType refers to a BaseType with baseTypeDefinition.baseTypeEncoding equal to UTF-8

- ImplementationDataType
  - with category ARRAY
  - that contains exactly one ImplementationDataTypeElement that boils down to a uint8 ImplementationDataType:
    * ImplementationDataTypeElement.arraySize is set to a value
    * ImplementationDataTypeElement.arraySizeSemantics is set to fixedSize

⌋

**[SWS_SomeIpXf_00241] Recognition of UTF-16 Fixed Length Strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈An UTF-16 Fixed Length String shall be detected if an ApplicationPrimitiveDataType and an ImplementationDataType with the following pattern are used:

- ApplicationPrimitiveDataType
  - with category equal to STRING
  - ApplicationPrimitiveDataType.swDataDefProps.swTextProps.baseType refers to a BaseType with baseTypeDefinition.baseTypeEncoding equal to UTF-16
- ImplementationDataType
  - with category ARRAY
  - that contains exactly one ImplementationDataTypeElement that boils down to a uint16 ImplementationDataType:
    * ImplementationDataTypeElement.arraySize is set to a value
    * ImplementationDataTypeElement.arraySizeSemantics is set to fixedSize

⌋

**[SWS_SomeIpXf_00321] Signal-Level String Length Field Insertion**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute sizeOfStringLengthFields of SOMEIPTransformationISignalProps is set to a value greater than 0, then the length field shall be inserted in front of every serialized string.⌋

**[SWS_SomeIpXf_00322] Transformation-Level String Length Field Insertion**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfStringLengthField` of `SOMEIPTransformationProps` is set to a value greater than 0, then the length field shall be inserted in front of the serialized string for which the SOMEIPTransformationProps is defined. (See [TPS_SYST_02121])⌋

**Note:**
`SOMEIPTransformationProps.sizeOfStringLengthField` and `SOMEIP-TransformationISignalProps.sizeOfStringLengthFields` have the same meaning but are defined at different modeling levels. They must not be set simultaneously ([8, System Template]constr_5246).

**[SWS_SomeIpXf_00323] Length Field Type Determination for Static Size Strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈If `SOMEIPTransformationProps.sizeOfStringLengthField` is present for a static size string, the data type for the length field of the string shall be determined by the value of `SOMEIPTransformationProps.sizeOfStringLengthField`:

- uint8 if sizeOfStringLengthField equals 1
- uint16 if sizeOfStringLengthField equals 2
- uint32 if sizeOfStringLengthField equals 4
- Otherwise [SWS_SomeIpXf_00324] applies.

⌋

**[SWS_SomeIpXf_00324] Length Field Type Determination for Serialized ISignals**

*Upstream requirements:* SRS_Xfrm_00101

⌈The data type of the length field for a string shall be determined by the value of `SOMEIPTransformationISignalProps.sizeOfStringLengthFields` of the serialized `ISignal` :

- uint8 if sizeOfStringLengthFields equals 1
- uint16 if sizeOfStringLengthFields equals 2
- uint32 if sizeOfStringLengthFields equals 4

⌋

**[SWS_SomeIpXf_00325] Serialized String Payload Size Specification**

*Upstream requirements:* SRS_Xfrm_00101

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

**[SWS_SomeIpXf_00244] Serialization of fixed length strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈Serialization of fixed length strings shall consist of the following steps:

1. Check whether the string terminates with `0x00` (`UTF-8`) or `0x0000` (`UTF-16`) and is not larger than the configured strings fixed length.If not, a `E_SER_GENERIC_ERROR` error shall be issued.
2. Add the Length Field - The value of the length field shall be computed by considering the number of elements given by the size indicator and the size in bytes of each element obtained during encoding of the Unicode codepoints into the respective transformation format (e.g., 1 up to 4 bytes for `UTF-8` and 2 or 4 bytes for `UTF-16`) increased by the size in bytes required by the `BOM` and Termination. The data type of the length field shall be determined from the `sizeOfStringLengthFields`. If the attribute `sizeOfStringLengthFields` is not configured explicitly, then the default value for `SIZE_OF_STRING_LENGTH_FIELD` as defined by [PRS_SOMEIP_00760] shall be used. The value of the length field shall comply with [CP_SWS_SomeIpXf_CONSTR_00001].
3. Append `BOM` at the beginning of the output buffer in the first three (`UTF-8`) or two (`UTF-16`) bytes of the to be serialized array containing the string.
4. Copying the string data (the number of bytes of the string to be serialized) from the array into the output buffer, optionally performing a conversion between `UTF-16LE` and `UTF-16BE` between ECU and network byte order if `BaseTypeDirectDefinition.byteOrder` and `SOMEIPTransformationDescription.byteOrder` have different values.
5. If the string content is shorter than the fixed length defined in the data type, the transformer shall pad the remaining bytes with the `NUL` character (`0x00` for `UTF-8`, `0x0000` for `UTF-16`) until the fixed length is reached (according to [PRS_SOMEIP_00951]).
   The number of bytes placed on the wire shall always equal the fixed length defined in the data type definition.

⌋

#### 7.1.4.4.2 Strings (dynamic length)

Strings with dynamic length can be realized in an AUTOSAR system as an array with dynamic length that transports the single characters.

**[SWS_SomeIpXf_00242] Recognition of UTF-8 Variable Length Strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈An UTF-8 Variable Length String shall be detected if an `ApplicationPrimitiveDataType` and an `ImplementationDataType` with the following pattern are used:

- ApplicationPrimitiveDataType
  - with category equal to STRING
  - ApplicationPrimitiveDataType.swDataDefProps.swTextProps.baseType refers to a BaseType with baseTypeDefinition.baseTypeEncoding equal to UTF-8
- ImplementationDataType
  The ImplementationDataType shall be defined according to [TPS_SWCT_01650] as a STRUCTURE that contains exactly two ImplementationDataTypeElements and shall follow the rules defined by [constr_1318]:
  - one ImplementationDataTypeElement represents the Size Indicator and has the category equal to TYPE_REFERENCE which points to a uint8, uint16 or uint32 ImplementationDataType
  - one ImplementationDataTypeElement has the category equal to ARRAY and contains exactly one ImplementationDataTypeElement that boils down to a uint8 ImplementationDataType

⌋

**[SWS_SomeIpXf_00243] Recognition of UTF-16 Variable Length Strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈An UTF-16 Fixed Length String shall be detected if an ApplicationPrimitiveDataType and an ImplementationDataType with the following pattern are used:

- ApplicationPrimitiveDataType
  - with category equal to STRING
  - ApplicationPrimitiveDataType.swDataDefProps.swTextProps.baseType refers to a BaseType with baseTypeDefinition.baseTypeEncoding equal to UTF-16
- ImplementationDataType
  The ImplementationDataType shall be defined according to [TPS_SWCT_01650] as a STRUCTURE that contains exactly two ImplementationDataTypeElements and shall follow the rules defined by [constr_1318]:
  - one ImplementationDataTypeElement represents the Size Indicator and has the category equal to TYPE_REFERENCE which points to a uint8, uint16 or uint32 ImplementationDataType
  - one ImplementationDataTypeElement has the category equal to ARRAY and contains exactly one ImplementationDataTypeElement that boils down to a uint16 ImplementationDataType

⌋

**[SWS_SomeIpXf_00245] Serialization of dynamic length strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈Serialization of dynamic length strings shall consist of the following steps:

1. Check whether the string terminates with `0x00` (UTF-8) or `0x0000` (UTF-16). If not, a E_SER_GENERIC_ERROR error shall be issued.
2. Add the Length Field - The value of the length field shall be computed by considering the number of elements given by the size indicator and the size in bytes of each element obtained during encoding of the Unicode codepoints into the respective transformation format (e.g., 1 up to 4 bytes for UTF-8 and 2 or 4 bytes for UTF-16) increased by the size in bytes required by the BOM and Termination. The data type of the length field shall be determined from the `sizeOfStringLengthFields`. If the attribute `sizeOfStringLengthFields` is not configured explicitly, then the default value for `SIZE_OF_STRING_LENGTH_FIELD` as defined by [PRS_SOMEIP_00002] shall be used.The value of the length field shall comply with [CP_SWS_SomeIpXf_CONSTR_00001].
3. Appending BOM at the beginning, 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
4. Copying the string data (copy the the number of bytes according to the string's size indicator and the size of bytes of each element) from the array into the output buffer, optionally performing a conversion between UTF-16LE and UTF-16BE between ECU and network byte order `BaseTypeDirectDefinition.byteOrder` and `SOMEIPTransformationDescription.byteOrder` have different values

⌋

#### 7.1.4.5 Arrays (fixed length)

**[SWS_SomeIpXf_00069]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The length of fixed length arrays is defined by the datatype definition.⌋

They can be seen as repeated elements. In chapter 7.1.4.7 dynamic length arrays are shown, which can be also used. Fixed length arrays are easier for use in very small devices. Dynamic length arrays might need more resources on the ECU using them.

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

**[SWS_SomeIpXf_00220]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfArrayLengthFields` of `SOMEIPTransformationISignalProps` is set to a value greater 0, a length field shall be inserted in front of every serialized array.⌋

**Note:**
[SWS_SomeIpXf_00220] also applies to nested arrays which means that additionally every nested fixed-size array has its own length field.

**[SWS_SomeIpXf_00256]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfArrayLengthField` of `SOMEIPTransformationProps` is set to a value greater 0, a length field shall be inserted in front of the serialized array for which the `SOMEIPTransformationProps` is defined. (See [TPS_SYST_02121])⌋

**Note:**
[SWS_SomeIpXf_00256] applies if the length fields of the array and all nested arrays contained are configured to different values for the lengths of the length fields via SOMEIPTransformationProps

**[SWS_SomeIpXf_00257]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If `SOMEIPTransformationProps.sizeOfArrayLengthField` is present for a static size array the data type for the length field of the array shall be determined by the value of `SOMEIPTransformationProps.sizeOfArrayLengthField`:

- *uint8* if `sizeOfArrayLengthField` equals 1
- *uint16* if `sizeOfArrayLengthField` equals 2
- *uint32* if `sizeOfArrayLengthField` equals 4
- Otherwise [SWS_SomeIpXf_00221] applies.

⌋

**[SWS_SomeIpXf_00320] Default for sizeOfArrayLengthFields**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfArrayLengthFields` of `SOMEIPTransformationISignalProps` is not configured explicitly, the default value for `SIZE_OF_ARRAY_LENGTH_FIELD` defined in [PRS_SOMEIP_00944] shall be used.⌋

**[SWS_SomeIpXf_00221]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The data type of the length field for an array shall be determined by the value of `SOMEIPTransformationISignalProps.sizeOfArrayLengthFields` of the serialized `ISignal`:

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

⌋

**[SWS_SomeIpXf_00222]**

*Upstream requirements:* SRS_Xfrm_00101

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

#### 7.1.4.5.1 One-dimensional

The one-dimensional arrays with fixed length n carry exactly n elements of the same type. The layout is shown in Figure 7.7.

**[SWS_SomeIpXf_00070]**

*Upstream requirements:* SRS_Xfrm_00101

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

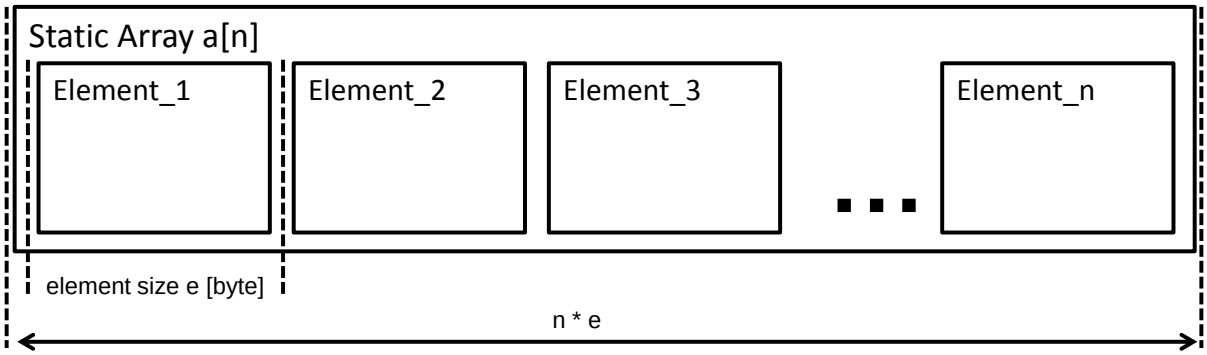


**Figure 7.7: One-dimensional array (fixed length)**

#### 7.1.4.5.2 Multidimensional

**[SWS_SomeIpXf_00072]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The serialization of multidimensional arrays shall happen in row-major order(in-memory layout of multidimensional arrays in the C programming language)⌋

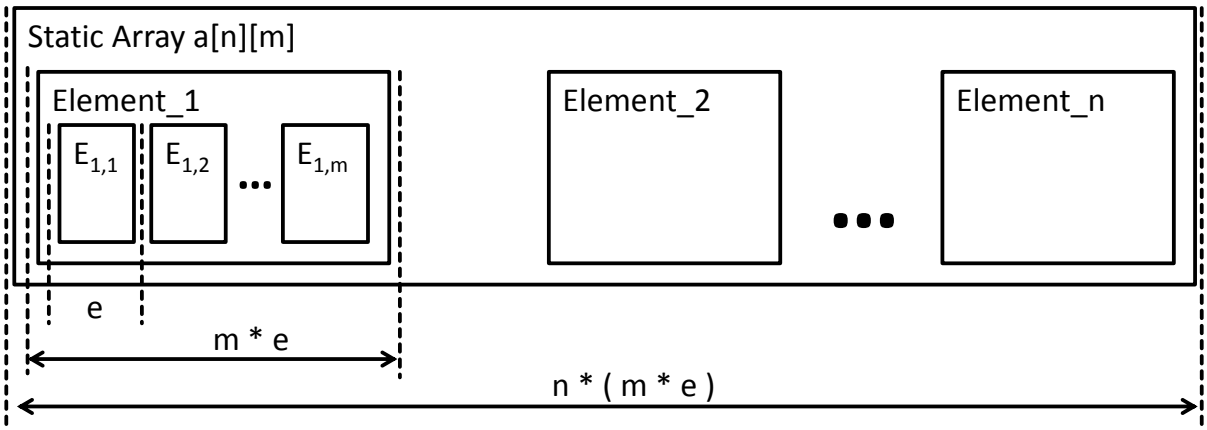


**Figure 7.8: Multidimensional array (fixed length)**

Consult [5] Chapter 5.3.4.4 *"Array Implementation Data Type"* for Arrays.

### 7.1.4.6 Optional Parameters / Optional Elements

Optional Elements can be encoded as array with 0 to 1 elements. For the serialization of arrays with dynamic length see Chapter 7.1.4.7.

### 7.1.4.7 Dynamic Length Arrays / Variable Size Arrays

Variable size arrays are implemented in AUTOSAR as structs with two members

- a size indicator which is an integer and holds the number of valid elements in the array
- the array with variable size

In SOME/IP variable size arrays are implemented in a similar manner. Only the size indicator is replaced by a length indicator.

- a length indicator which is an integer and holds the length (in bytes) of the following variable size array
- the array which contains the valid elements of the variable size array

In AUTOSAR also so called "old-world" variable-size array data types exist which don't have a size indicator. These are not supported by data transformation in general and

hence also not supported by the SOME/IP transformer. For details, refer to [constr_1387] ([8, System Template]), [TPS_SWCT_01644], [TPS_SWCT_01645] and [TPS_SWCT_01642].

**[SWS_SomeIpXf_00076]**

*Upstream requirements:* SRS_Xfrm_00101

⌈A variable size array embedded in a structure which also contains a size indicator shall be serialized as the concatenation of the following elements:

- the length indicator which holds the length (in bytes) of the following variable size array
- the array which contains the valid elements of the variable size array

where

- the data type of the length field shall be determined as specified in [SWS_SomeIpXf_00234]
- the array shall be serialized like a static size array but does only contain the valid elements. The number of elements to serializer shall be taken from the size indicator.

⌋

**[SWS_SomeIpXf_00234]**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00008

⌈A variable size array is represented in AUTOSAR by an `ImplementationDataType` with the category `STRUCTURE` and two sub-elements (namely *payload* and *size indicator*). The data type of the length fields for the SOME/IP message for an variable size array shall be determined from the `sizeOfArrayLengthFields`. If the attribute `sizeOfArrayLengthFields` is not configured then the default value for `SIZE_OF_ARRAY_LENGTH_FIELD` as defined by [PRS_SOMEIP_00001] shall be used. In case of nested variable size arrays, AUTOSAR allows to use profiles to specify size indicators which apply to more than one variable size array nested within the same `ImplementationDataType`. Depending on the specific profile (`dynamicArraySizeProfile`), the data type of the of the length fields inside the SOME/IP message shall be determined differently:

- `VSA_LINEAR`
  The data type of the SOME/IP length field shall be determined from the single `sizeOfArrayLengthFields`. If the attribute `sizeOfArrayLengthFields` is not configured then the default value for `SIZE_OF_ARRAY_LENGTH_FIELD` as defined by [PRS_SOMEIP_00001] shall be used.
- `VSA_SQUARE`
  All data type of the SOME/IP length fields shall be determined from the single `sizeOfArrayLengthFields`. If the attribute `sizeOfArrayLengthFields` is

not configured then the default value for SIZE_OF_ARRAY_LENGTH_FIELD as defined by [PRS_SOMEIP_00001] shall be used.

- VSA_RECTANGULAR
  The data type of all SOME/IP length fields for all dimensions (nesting level) shall be determined from the single sizeOfArrayLengthFields. If the attribute sizeOfArrayLengthFields is not configured then the default value for SIZE_OF_ARRAY_LENGTH_FIELD as defined by [PRS_SOMEIP_00001] shall be used.
- VSA_FULLY_FLEXIBLE
  The data type of all SOME/IP length fields for all variable size arrays shall be determined from the single sizeOfArrayLengthFields. If the attribute sizeOfArrayLengthFields is not configured then the default value for SIZE_OF_ARRAY_LENGTH_FIELD as defined by [PRS_SOMEIP_00001] shall be used.

⌋

This means only the first m elements of the variable size array are serialized where m is the value of the size indicator.

The layout of dynamic arrays is shown in 7.9 and Figure 7.10 where L_1 and L_2 denote the length in bytes.

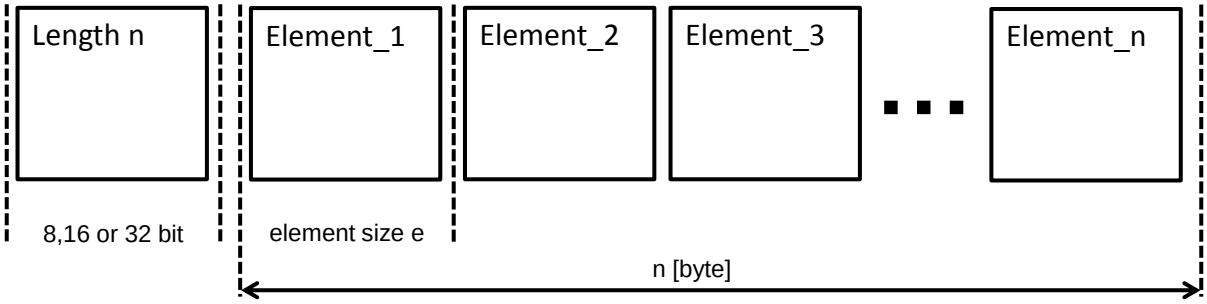


**Figure 7.9: One-dimensional array (dynamic length) (Example)**

In the one-dimensional array one length field is used, which carries the size in bytes of the valid elements in the array.

**[SWS_SomeIpXf_00235]**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00008

⌈If the value of dynamicArraySizeProfile equals VSA_LINEAR, the value of the length field of the serialized variable size array shall be calculated based on the value of the size indicator of the AUTOSAR data type.⌋

The number of static length elements can be easily calculated by dividing the array length n by the Byte size of an element.

In the case of dynamical length elements the number of elements cannot be calculated but the elements must be parsed sequentially.

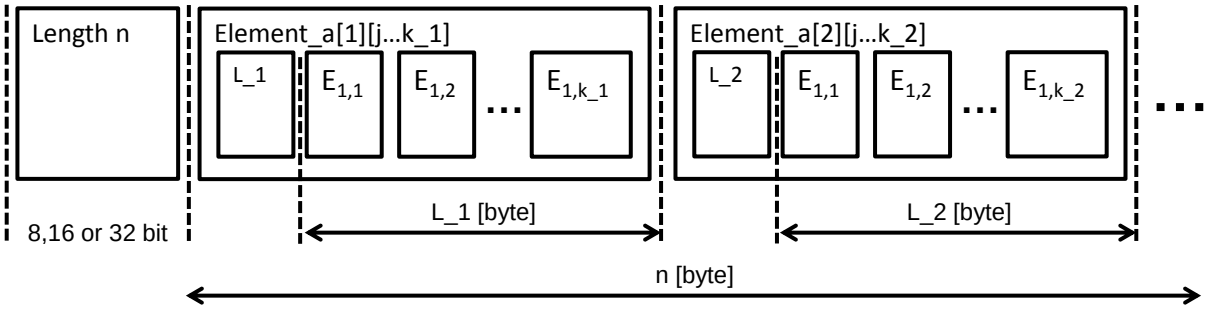


**Figure 7.10: Multidimensional array (dynamic length) (Example)**

In case of multidimensional variable size arrays, each variable size array needs to have its own length field, independent of the way how the variable size array is designed in the AUTOSAR data type (i.e. independent from the value of `dynamicArraySizeProfile`) as specified in [SWS_SomeIpXf_00234]. Hence it is supported to have different length columns and different length rows in the same dimension. See `k_1` and `k_2` in Figure 7.10.

**[SWS_SomeIpXf_00236]**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00008

⌈If the value of `dynamicArraySizeProfile` of a multi-dimensional variable size array equals `VSA_SQUARE`, the value of all length fields of the nested serialized variable size arrays that belong to this multi-dimensional variable size arrays shall be calculated based on the value of the single size indicator of the AUTOSAR data type.⌋

In case of `VSA_SQUARE`, the AUTOSAR data type only has one size indicator. The value of this size indicator will be used as base for the calculation for the value of all length fields of such a multi-dimensional variable size array.

**[SWS_SomeIpXf_00237]**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00008

⌈If the value of `dynamicArraySizeProfile` of a multi-dimensional variable size array equals `VSA_RECTANGULAR`, the values of all length fields of the nested serialized variable size arrays of the same nesting level (i.e. the same dimension) that belong to this multi-dimensional variable size array shall be calculated based on the values of the size indicators of the AUTOSAR data type for this respective nesting level.⌋

In case of `VSA_RECTANGULAR`, the AUTOSAR data type has exactly one size indicator for each dimension of the the multi-dimensional variable size array. For all variable size arrays in one dimension, the value of the according size indicator of this dimension will be used as base for the calculation of the values of all length fields of this dimension.

**[SWS_SomeIpXf_00238]**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00008

⌈If the value of `dynamicArraySizeProfile` of a multi-dimensional variable size array equals `VSA_FULLY_FLEXIBLE`, the values of all length fields of the nested serial-

ized variable size arrays that belong to this multi-dimensional variable size arrays shall be calculated based on the value of the size indicator of the corresponding variable size array that is contained in the AUTOSAR data type.⌋

In case of VSA_FULLY_FLEXIBLE, in the AUTOSAR data type the outer variable size array and each nested variable size arrays has its own size indicator. For the calculation of the values of the length fields both of the outer and all nested variable size arrays the according values of the size indicators of the AUTOSAR data type will be used as base.

The RTE provides a buffer where serialization result will be written into by the SOME/IP transformer which is large enough to keep the length field and a fully filled dynamic array.

**[SWS_SomeIpXf_00326] Maximum number of array elements**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00008

⌈The maximum number of variable size array elements shall be defined by the ImplementationDataTypeElement.arraySize attribute of the respective ImplementationDataTypeElement depending on the ImplementationDataType.dynamicArraySizeProfile (see [constr_1318], [constr_1319], [constr_1320], and [constr_1321]).⌋

#### 7.1.4.8 Bitfield

**[SWS_SomeIpXf_00300]**

*Upstream requirements:* SRS_Xfrm_00101

⌈Bitfields shall be transported as-is based on the underlying SwBaseType uint8/uint16/uint32/uint64 according to [SWS_SomeIpXf_00036]. No further modification or interpretation shall be done by the SOME/IP transformer.⌋

#### 7.1.4.9 Union / Variant

A union (also called variant) is a parameter that can contain different types of elements. For example, if one defines a union of type uint8 and type uint16, the union shall carry an element of uint8 or uint16.

The union serialization will only be triggered if the pattern defined in [SWS_SomeIpXf_00249] applies.

**[SWS_SomeIpXf_00249]**

*Upstream requirements:* SRS_Xfrm_00101

⌈A union shall be detected if an `ImplementationDataType` with the following pattern (named wrapped union data type) is used: `ImplementationDataType` with category `STRUCTURE` that contains exactly two `ImplementationDataTypeElement`s:

- memberSelector: `ImplementationDataTypeElement` which represents the type field that boils down to a uint8, uint16 or uint32 `ImplementationDataType`
- payload: `ImplementationDataTypeElement` of category `UNION` which represents the actual union

⌋

When using different types of elements the alignment of subsequent parameters may be distorted. To resolve this, padding might be needed.

**[SWS_SomeIpXf_00088] Default serialization layout of unions in SOME/IP**

*Upstream requirements:* SRS_Xfrm_00101

⌈

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

⌋

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

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

**[SWS_SomeIpXf_00224]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfUnionLengthFields` of `SOMEIPTransformationISignalProps` is set to a value greater 0, a length field shall be inserted in front of every serialized union.⌋

**Note:**
[SWS_SomeIpXf_00224] also applies to nested unions which means that additionally every nested union has its own length field.

**[SWS_SomeIpXf_00254]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If attribute `sizeOfUnionLengthField` of `SOMEIPTransformationProps` is set to a value greater 0, a length field shall be inserted in front of the serialized union for which the `SOMEIPTransformationProps` is defined. (See [TPS_SYST_02121]).⌋

**Note:**
[SWS_SomeIpXf_00254] applies if the length fields of the union and all nested unions contained within the root union are configured to different values for the lengths of the length fields via `SOMEIPTransformationProps`.

**[SWS_SomeIpXf_00225]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The data type of the length field of the union and all nested unions within the union shall be determined by the value of `SOMEIPTransformationISignalProps.sizeOfUnionLengthFields` of the serialized `ISignal`:

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

⌋

**[SWS_SomeIpXf_00258]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If `SOMEIPTransformationProps.sizeOfUnionLengthField` is present for a union the data type of the length field for the union shall be determined by the value of `SOMEIPTransformationProps.sizeOfUnionLengthField`:

- *uint8* if `sizeOfUnionLengthFields` equals 1
- *uint16* if `sizeOfUnionLengthFields` equals 2
- *uint32* if `sizeOfUnionLengthFields` equals 4
- If `SOMEIPTransformationProps.sizeOfUnionLengthField` is not configured explicitly, the default value as defined for `SIZE_OF_UNION_LENGTH_FIELD` in [PRS_SOMEIP_00121] shall be used.
- Otherwise [SWS_SomeIpXf_00225] applies.

⌋

**[SWS_SomeIpXf_00226]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The serializing SOME/IP transformer shall write the size (in bytes) of the serialized union (including padding bytes but without the size of the length field and type field)

into the length field of the union. This requirement does not apply for the serialization of extensible structs and methods.⌋

**Note:**
See also chapter 7.1.4.3.

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

For length of the type field see [PRS_SOMEIP_00127].

The type field describes the type of the element.

**[SWS_SomeIpXf_00250]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The data type of the type field of the union shall be determined from the `ImplementationDataType` of the first `ImplementationDataTypeElement` (memberSelector) in the wrapped union data type defined in [SWS_SomeIpXf_00249].⌋

**[SWS_SomeIpXf_00098]**

*Upstream requirements:* SRS_Xfrm_00101

⌈Possible values of the type field are defined by the data type specification of the union. The types are encoded as in the data type in ascending order starting with 1. The 0 is reserved for the NULL type - i.e. an empty union.⌋

**[SWS_SomeIpXf_00251]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The value of the type field shall be set to the value defined by the first `ImplementationDataTypeElement` (memberSelector) in the wrapped union data type defined in [SWS_SomeIpXf_00249].⌋

**[SWS_SomeIpXf_00099]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The element is serialized depending on the type in the type field. This also defines the length of the data. All bytes behind the data that are covered by the length, are padding. The deserializer shall skip the padding bytes by calculating the required number according to the formula given in [SWS_SomeIpXf_00088].⌋

By using a struct in the data type definition, different padding layouts can be achieved.

#### 7.1.4.9.1 Example: Union of uint8/uint16 both padded to 32 bit

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

A uint8 will be serialized like this:

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

A uint16 will be serialized like this:

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

### 7.1.5 De-serialization of Parameters and Data Structures

The de-serialization process need to inspect the payload (serialized byte stream) of the received SOME/IP message. Thereby the de-serialization process need to identify the elements within the received byte stream and compare the identified elements with the configured data type(s) of the corresponding service interface (please note, the data type is derived from the interface specification, which defines the exact position of all data structures in a PDU). The possibility to identify elements in a dedicated SOME/IP serialized byte stream depend on the interface specification and the serialization properties. The serialization properties define among others:

- if structured data types are serialized with a length field in front
- if tag-length-value are used for encoding, which include data ids and the possibility specify optional data members

The de-serialization process of a SOME/IP messages need to consider the received message length and deal with a message length which may be larger than expected according the interface specification. This is needed to support backward compatible communication, where ECUs of a heterogeneous in-vehicle network (re-used ECUs and new developed ECUs) communicate via SOME/IP serialized byte streams. Note that the feature of "complementary default value during reception of less data than expected" is no longer supported by AUTOSAR. The subsequential chapters describe the expected behavior of the de-serialization process.

**[SWS_SomeIpXf_00311] De-serialization - `SenderReceiverInterface` or `ClientServerInterface`**

*Upstream requirements:* SRS_Xfrm_00101

⌈The de-serialization shall consider the `SenderReceiverInterface` or `ClientServerInterface` of the data which is de-serialized.⌋

**[SWS_SomeIpXf_00169]**

*Upstream requirements:* SRS_Xfrm_00101

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

This means: Parameters that were not defined in the `ClientServerInterface` or `SenderReceiverInterface` used to generate or parameterize the deserialization code at the end of the serialized data will be ignored by the deserialization.

**[SWS_SomeIpXf_00016]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If more data than expected are handed over to the SOME/IP transformer during deserialization of data, the unexpected data shall be discarded. The known fraction shall be considered.⌋

**[SWS_SomeIpXf_00017]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If less data than expected are handed over to the SOME/IP transformer during deserialization of data, then abort deserialization with `E_SER_MALFORMED_MESSAGE`.⌋

#### 7.1.5.1 Structured Datatypes (structs)

**[SWS_SomeIpXf_00219]**

*Upstream requirements:* SRS_Xfrm_00101

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

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

#### 7.1.5.2 Structured Datatypes and Arguments with Identifier and optional Members

**[SWS_SomeIpXf_00294]**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the transformer cannot find a required (i.e. non-optional) member defined in its data definition in the serialized byte stream, the deserialization shall be aborted with `E_SER_MALFORMED_MESSAGE`. For examples, please refer to [7].⌋

**[SWS_SomeIpXf_00314] Deserialization with invalid wire type**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the transformer finds a required (i.e. non-optional) member defined in its data definition in the serialized byte stream with an invalid wire type, then the deserialization shall be aborted with `E_SER_MALFORMED_MESSAGE`.⌋

**[SWS_SomeIpXf_00315] Deserialization with duplicate members**

*Upstream requirements:* SRS_Xfrm_00106

⌈If the transformer finds a member/argument defined in its data definition in the serialized byte stream multiple times, then the de-serialization shall be aborted with `E_SER_MALFORMED_MESSAGE`.⌋

#### 7.1.5.3 Strings

##### 7.1.5.3.1 Strings (fixed length)

**[SWS_SomeIpXf_00246] Deserialization of fixed length strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈Deserialization of fixed length strings shall consist of the following steps:

1. Check whether the string starts with a `BOM`. If not, a `MALFORMED_MESSAGE` error shall be issued
2. Check whether `BOM` has the same value as `SOMEIPTransformationDescription.byteOrder`. If not, a `MALFORMED_MESSAGE` error shall be issued
3. Remove the `BOM`
4. Silently discard the last byte of the string in case of an `UTF-16` string with odd length
5. Check whether the string terminates with `0x00` (`UTF-8`) or `0x0000` (`UTF-16`). If not, a `MALFORMED_MESSAGE` error shall be issued
6. Copy the string data (the number of bytes according to the string's fixed length) from the input buffer into the array, optionally performing a conversion between `UTF-16LE` and `UTF-16BE` between network and ECU byte order if `BaseTypeDirectDefinition.byteOrder` and `SOMEIPTransformationDescription.byteOrder` have different values.
7. If a `NUL` terminator is present anywhere within the fixed-length of string, the string shall be accepted even if its actual length is shorter than the fixed length.
8. The message shall be treated as malformed with a `MALFORMED_MESSAGE` error if:

- No `NUL` terminator appears within the fixed-length string, or
- A length field (if present) exceeds the fixed-length defined in the data type.

⌋

#### 7.1.5.3.2 Strings (dynamic length)

**[SWS_SomeIpXf_00247] Deserialization of dynamic length strings**

*Upstream requirements:* SRS_Xfrm_00101

⌈Deserialization of dynamic length strings shall consist of the following steps:

1. Check whether the string starts with a BOM. If not, a `MALFORMED_MESSAGE` error shall be issued
2. Check whether BOM has the same value as `SOMEIPTransformationDescription.byteOrder`. If not, a `MALFORMED_MESSAGE` error shall be issued
3. Remove the BOM and reduce the value of the length field accordingly
4. Silently discard the last byte of the string in case of an UTF-16 string with odd length (according to the reduced value of the length field)
5. Check whether the string terminates with `0x00` (UTF-8) or `0x0000` (UTF-16). If not, a `MALFORMED_MESSAGE` error shall be issued
6. Check whether the length of the received dynamic length string is less or equal than the specified maximum length of the string (ApplicationPrimitiveDataType.swTextProps.swMaxTextSize or arraySize of ImplementationDataTypeElement of category ARRAY). If not, a MALFORMED_MESSAGE error shall be issued.
7. Copy the string data (copy the number of bytes according to the string's reduced value of the length field) from the input buffer into the array, optionally performing a conversion between (UTF-16LE) and (UTF-16BE) between ECU and bus if `BaseTypeDirectDefinition.byteOrder` and `SOMEIPTransformationDescription.byteOrder` have different values.

⌋

Instead of transferring application strings as SOME/IP strings with BOM and "\0" termination, strings can also be transported as plain dynamic length arrays without BOM and "\0" termination (see chapter *Dynamic Length Arrays* of [13]).

#### 7.1.5.4 Arrays (fixed length)

**[SWS_SomeIpXf_00223]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If the length is greater than the expected length of an array (as specified in the data type definition) a deserializing SOME/IP transformer shall only interpret the expected data and skip the unexpected. Additionally, an error E_SER_PAYLOAD_LENGTH_EXCEEDED (see also [SWS_Xfrm_00031]) shall be issued.⌋

**Note:** This does not necessarily mean that the message needs to be dropped.

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

#### 7.1.5.5 Dynamic Length Arrays / Variable Size Arrays

No further requirements considered for the deserialization.

#### 7.1.5.6 Bitfield

No further requirements considered for the deserialization.

#### 7.1.5.7 Union / Variant

**[SWS_SomeIpXf_00227]**

*Upstream requirements:* SRS_Xfrm_00101

⌈If the length is greater than the expected length of a union (as specified in the data type definition) a deserializing SOME/IP transformer shall only interpret the expected data and skip the unexpected.⌋

Please consider [SWS_SomeIpXf_00099] for skipping padding bytes of serialized unions / variant within the de-serialization process.

## 7.2 Protocol specification

This chapter describes the protocol of SOME/IP for Client/Server and Sender/Receiver communication.

### 7.2.1 Client/Server Communication

**[SWS_SomeIpXf_00106]**

*Upstream requirements:* SRS_Xfrm_00102

⌈For the SOME/IP request message, the SOME/IP transformer on the client-ECU has to do the following for payload and header:

- Construct the payload
- Optionally set the Request ID to a unique number (shall be unique for client only)
- Set the Protocol Version according [SWS_SomeIpXf_00029]
- Set the Interface Version. If `interfaceVersion` of `SOMEIPTransformationISignalProps` is set, this shall be used. Otherwise `interfaceVersion` of `SOMEIPTransformationDescription` shall be used.
- Set the Message Type to Request (i.e. `0x00`)
- Set the Return Code to `0x00`

⌋

**[SWS_SomeIpXf_00120]**

*Upstream requirements:* SRS_Xfrm_00102

⌈To construct the payload of a request message all `argument`s of the `ClientServerOperation` which have `direction` `IN` or `INOUT` shall be serialized according to the order of the `ArgumentDataPrototype`s within the `ClientServerOperation`.⌋

This can be seen graphically in Figure 7.11.

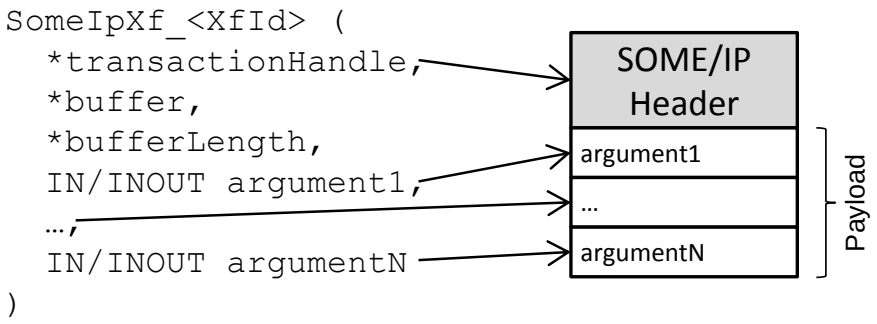


**Figure 7.11: Example for serialization of a Client/Server Request**

**[SWS_SomeIpXf_00200]**

*Upstream requirements:* SRS_Xfrm_00102

⌈If `csErrorReaction` of `TransformationISignalProps` is set to `autonomous` and the `returnValue` parameter handed over from RTE is greater or equal to `0x80`,

the SOME/IP transformer for a response of a client/server communication shall generate an error message according to [SWS_SomeIpXf_00201]. If csErrorReaction of TransformationISignalProps is set to autonomous and the returnValue parameter handed over from RTE is lesser than 0x80,the SOME/IP transformer shall generate a normal response according to [SWS_SomeIpXf_00107].⌋

**[SWS_SomeIpXf_00107]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The SOME/IP transformer on the server-ECU builds its header for the server response based on the header of the client's request and does in addition:

- Construct the payload
- Set the Message Type to RESPONSE (i.e. 0x80)
- If the ClientServerOperation has at least one possibleError defined, place the return value of the executed ClientServerOperation into the Return Code field and add 0x1F to adapt the number ranges in case the original return value was different from 0x00.

⌋

**Note:** See also chapter 7.1.3.5.

**[SWS_SomeIpXf_00121]**

*Upstream requirements:* SRS_Xfrm_00102

⌈To construct the payload of a response message all arguments of the ClientServerOperation which have direction INOUT or OUT shall be serialized in the following order:
The ArgumentDataPrototypes with a direction of INOUT or OUT shall be serialized according to the order of the ArgumentDataPrototypes within the ClientServerOperation.⌋

This can be seen graphically in Figure 7.12.

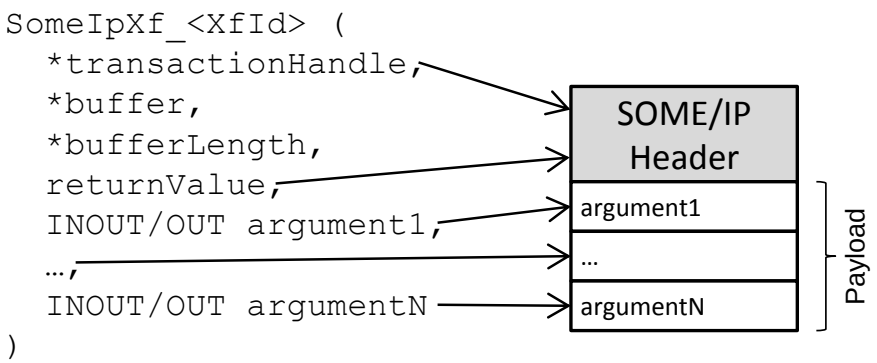


**Figure 7.12: Example for serialization of a Client/Server Response**

**[SWS_SomeIpXf_00201]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The SOME/IP transformer on the server-ECU builds its header for an autonomous error response based on the header of the client's request and does in addition:

- Construct no payload (the payload shall be empty)
- Set the Message Type to `RESPONSE` (i.e. `0x80`)
- Adapt the return value by subtracting `0x80` from the parameter `returnValue` (calculation: adaptedReturnValue = returnValue - 0x80)
- Place the `adaptedReturnValue` into the Return Code field.

⌋

**Note:** See also chapter 7.1.3.5.

This leads to an output of the SOME/IP transformer which is exactly as long as the SOME/IP header.

**Note:**
Error messages can only be sent as a response for client/server requests, not for Sender/Receiver communication or error messages.

**[SWS_SomeIpXf_00202]**

*Upstream requirements:* SRS_Xfrm_00102

⌈A SOME/IP transformer on the server-ECU that builds an autonomous error response shall return with a return value equal to `E_OK` (See [SWS_SomeIpXf_00141]).⌋

If the SOME/IP transformer would return with a return code different from `E_OK` this would issue a hard error that prevents the RTE from sending the autonomous error response.

**[SWS_SomeIpXf_00312] Return Code in case of an autonomous error response**

*Upstream requirements:* SRS_Xfrm_00102

⌈Upon reception of a SOME/IP RESPONSE message,the inverse SOME/IP transformer on the client-ECU shall additionally evaluate the Return Code field in case of an autonomous error response and do the following:
If the received SOME/IP RESPONSE message is an autonomous error response (i.e., the value of Return Code field is different from 0x00 (E_OK) and smaller than 0x20), then the inverse SOME/IP transformer on the client-ECU (SomeIpXf_Inv_<transformerId>()) shall not perform any de-serialization of the payload and shall use the value of the Return Code field incremented by 0x80 as its own return value (calculation: returnValue =valueOfReturnCodeField + 0x80).⌋

**Note:**This will lead to a hard error and proper signaling via the return value and (if configured) via the transformer Error OUT argument of Rte_Call() or Rte_Result() to the client application software component.

**[SWS_SomeIpXf_00313] Return Code in case of a Client-Server operation response**

*Upstream requirements:* SRS_Xfrm_00102

⌈Upon reception of a SOME/IP RESPONSE message,the inverse SOME/IP transformer on the client-ECU shall additionally evaluate the Return Code field in case of a Client-Server operation response and do the following:
If the received SOME/IP RESPONSE message is a response produced by the Client-Server operation (i.e., the value of Return Code field is either 0x00 (E_OK) or larger or equal to 0x20), then the inverse SOME/IP transformer on the client ECU (SomeIpXf_Inv_<transformerId>()) shall perform the de-serialization of the payload and shall hand over the value of the Return Code field (reduced by 0x1F in case its value is different from 0x00.⌋

**Note:**0x1F gets added on the server-ECU according to [SWS_SomeIpXf_00107] via the returnValue OUT argument.

### 7.2.2 Sender/Receiver Communication

Session Handling ID counter is used to set the correct Request ID in the SOME/IP header in case of Sender/Receiver communication where session handling is activated.

**[SWS_SomeIpXf_00212]**

*Upstream requirements:* SRS_Xfrm_00008

⌈One Session Handling ID counter (16 Bit) has to be maintained per transformer function for Sender/Receiver communication if the transmission path includes SomeIpTp.⌋

**[SWS_SomeIpXf_00213]**

*Upstream requirements:* SRS_Xfrm_00008

⌈All Session Handling ID counters shall be initialized with `0x0001`.⌋

**[SWS_SomeIpXf_00108]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The SOME/IP transformer on the sender side of transformed Sender/Receiver communication shall construct header and payload in the following way:

- Construct the payload
- Set the Request ID
  - to `0x00` if the transmission path does not include SomeIpTp
  - the current value of the Session Handling ID counter otherwise
- Set the Protocol Version according [SWS_SomeIpXf_00029]

- Set the Interface Version. If `interfaceVersion` of `SOMEIPTransformationISignalProps` is set, this shall be used. Otherwise `interfaceVersion` of `SOMEIPTransformationDescription` shall be used.
- Set the Message Type according to `messageType` of `SOMEIPTransformationISignalProps`:
  - `NOTIFICATION` (`0x02`) shall be used in the header if attribute `messageType` is set to `notification`
  - `REQUEST_NO_RETURN` (`0x01`) shall be used in the header if attribute `messageType` is set to `requestNoReturn`
- Set the Return Code to `0x00`

⌋

In [SWS_SomeIpXf_00108] it is specified when session handling is considered for messages which are sent. The SOME/IP transformer never checks the session ID on receiver side because the default behavior of SOME/IP is for sender/receiver communication to ignore session IDs on receiver side.

**[SWS_SomeIpXf_00176]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The payload of a message for Sender/Receiver communication shall consists of the serialized data element that is transported.⌋

Error handling and return codes have to be implemented by the application when needed.

### 7.2.3 External Trigger Events

External trigger events are used to trigger RPCs without any IN, INOUT or OUT arguments or to represent a special kind of an event without any parameters that is transmitted from a server to one or more client(s) and at which occurrence the Service Consumer shall react in a particular manner. External trigger events are realized by SOME/IP as fire-and-forget methods without arguments

**[SWS_SomeIpXf_00204]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The SOME/IP transformer on the trigger source side of transformed external trigger events shall construct header in the following way:

- Set the Request ID
  - to `0x00` if the transmission path does not include SomeIpTp

  - the current value of the Session Handling ID counter otherwise
- Set the Protocol Version according [SWS_SomeIpXf_00029]
- Set the Interface Version. If `interfaceVersion` of `SOMEIPTransformationISignalProps` is set, this shall be used. Otherwise `interfaceVersion` of `SOMEIPTransformationDescription` shall be used.
- Set the Message Type to `REQUEST_NO_RETURN` (i.e. `0x01`)
- Set the Return Code to `0x00`

⌋

**[SWS_SomeIpXf_00205]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The payload of a message for external trigger event communication shall be empty.⌋

Error handling and return codes have to be implemented by the application when needed.

### 7.2.4 Error Handling

The error handling will be done solely in the application. SOME/IP only transports the errors.

Two different mechanisms for error transportation are supported: Return Code and Error Message

**[SWS_SomeIpXf_00111]**

*Upstream requirements:* SRS_Xfrm_00102, SRS_Xfrm_00103, SRS_BSW_00331, SRS_BSW_00452, SRS_BSW_00458, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00481

⌈The SOME/IP transformer shall use the Return Code error handling, using Message Type RESPONSE (0x80) according to [PRS_SOMEIP_00901] when creating error responses. See [SWS_SomeIpXf_00201]⌋

**[SWS_SomeIpXf_00310] Handling of RESPONSE and ERROR Message Types**

*Upstream requirements:* SRS_Xfrm_00102, SRS_Xfrm_00103

⌈The SOME/IP transformer shall use the Return Code error Handling for Message Type RESPONSE(0x80) according to [PRS_SOMEIP_00901] when receiving error responses.

The SOME/IP transformer shall also handle responses of Message Type ERROR(0x81) according to [PRS_SOMEIP_00902] and [PRS_SOMEIP_00903] but without using the Payload of the Error Message, since this is not yet supported by this

version of the SOME/IP transformer. Only the Return Code value is used in this case. See [SWS_SomeIpXf_00149].⌋

Note: The reason to handle Message Type ERROR(0x81) responses is to be able to handle interoperability between AP and CP.

All messages have a return code field to carry the return code. However, only responses (Message Types 0x80 and 0x81) use this field to carry a return code to the request (Message Type 0x00) they answer. All other messages set this field to 0x00 (see Chapter 7.1.3.4)

#### 7.2.4.1 Return Code

**[SWS_SomeIpXf_00112]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The Error Handling via Return Code shall be based on the `Std_ReturnType`.⌋

**[SWS_SomeIpXf_00113]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The Return Codes shall only be used for Client/Server communication⌋

**[SWS_SomeIpXf_00170]**

*Upstream requirements:* SRS_Xfrm_00102

⌈In case of Client/Server communication the Return Code shall transport the `ApplicationError`s of the executed `ClientServerOperation` if no SOME/IP error occurred.⌋

This means: If a SOME/IP error occurred, this error is contained in the Return Code. If no SOME/IP error occurred, the Return Code contains the error (or success) code of the executed server runnable.

If an error occurs in case of client/server communication the server can be configured to create an autonomous error reaction which will be sent back to the client. In that response, the SOME/IP header fields `RequestId` and `Interface Version` shall be equal to the values in the header of the request message.

This is realized by [SWS_SomeIpXf_00201] which fills the header fields accordingly: `RequestId` is handed over from RTE and `InterfaceVersion` is consistent to the request as the configuration of the SOME/IP transformer only allows the same `interfaceVersion` for request and response.

**[SWS_SomeIpXf_00115] Return Codes**

*Upstream requirements:* SRS_Xfrm_00102, SRS_BSW_00170, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00310

⌈

| ID | Name | Description |
|---|---|---|
| 0x00 | E_OK | No error occurred |
| 0x01 | E_NOT_OK | An unspecified error occurred |
| 0x04 | SOMEIPXF_E_NOT_READY | deprecated. |
| 0x05 | SOMEIPXF_E_NOT_REACHABLE | deprecated. |
| 0x06 | SOMEIPXF_E_TIMEOUT | deprecated. |
| 0x07 | SOMEIPXF_E_WRONG_ PROTOCOL_VERSION | Version of SOME/IP protocol not supported |
| 0x08 | SOMEIPXF_E_WRONG_ INTERFACE_VERSION | Interface version mismatch |
| 0x09 | SOMEIPXF_E_ MALFORMED_MESSAGE | Deserialization error, so that payload cannot be deserialized. |
| 0x0a | SOMEIPXF_E_ WRONG_MESSAGE_TYPE | An unexpected message type was received.(e.g.received REQUEST for a method defined as REQUEST_NO_RETURN). |
| 0x0d | E_E2E | Not further specified E2E error |
| 0x0b - 0x1c<br>0x0e - 0x1f | RESERVED | Reserved for generic SOME/IP errors. These errors will be specified in future versions of this document. |
| 0x20 - 0x5e | - | Specific ApplicationErrors of ClientServerOperations. These errors are the application errors specified by the ClientServerInterface.<br>As the range of ApplicationErrors of the ClientServerInterface is 0x01-0x3F, the value of an ApplicationError has to be adapted for transport over SOME/IP by adding 0x1F. |

⌋

#### 7.2.4.2 Communication Errors and Handling of Communication Errors

When considering the transport of Client/Server messages different reliability semantics exist:

- Maybe — the message might reach the communication partner
- At least once — the message reaches the communication partner at least once
- Exactly once — the message reaches the communication partner exactly once

When using these terms in regard to client/server communication the term applies to both messages (i.e. call and response or error).

While different implementations may implement different approaches, SOME/IP transformer currently achieves "maybe" reliability when using the UDP binding and "exactly once" reliability when using the TCP binding by a suitable configuration of the Ethernet modules. Further error handling is left to the application.

For "maybe" reliability, only a single timeout is needed, when using client/server communication in combination with UDP as transport protocol. Figure 7.13 shows the state machines for "maybe" reliability. The client's SOME/IP implementation has to

wait for the response for a specified timeout. If the timeout occurs SOME/IP shall signal SOMEIPXF_E_TIMEOUT to the client application.

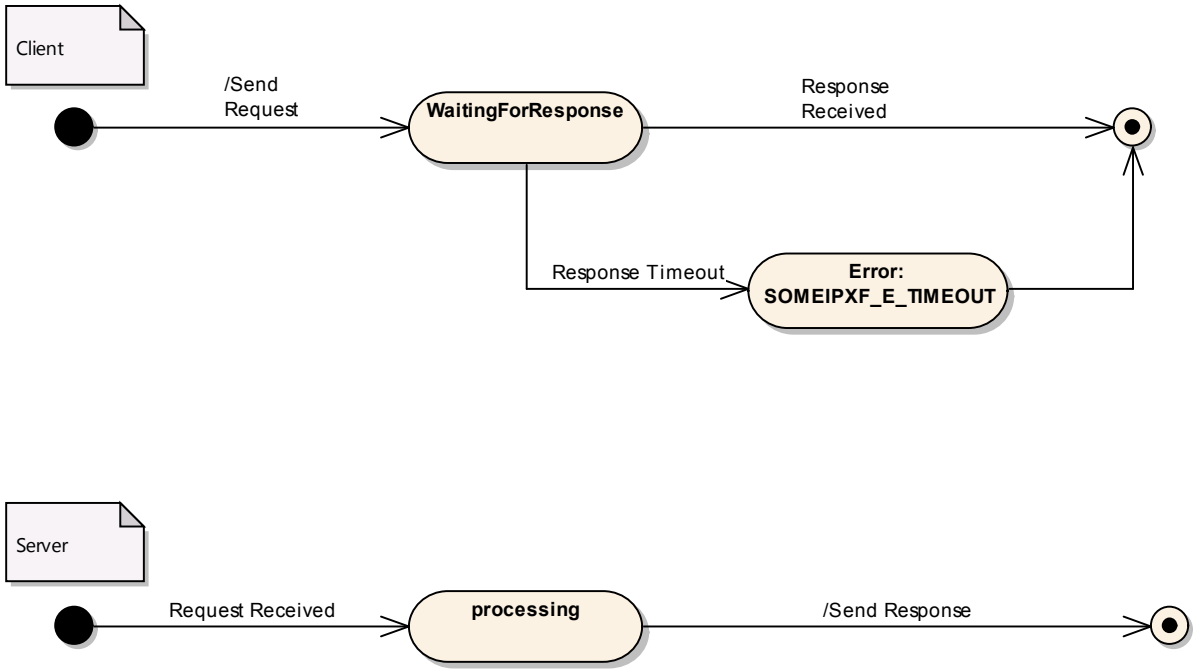


**Figure 7.13: State Machines for Reliability "Maybe"**

For "exactly once" reliability the TCP binding may be used, since TCP was defined to allow for reliable communication.

Additional mechanisms to reach higher reliability may be implemented in the application or in a SOME/IP implementation. Keep in mind that the communication does not have to implement these features. Chapter 7.2.4.2.1 describes such optional reliability mechanisms.

#### 7.2.4.2.1 Application based Error Handling

The application can easily implement "at least once" reliability by using idempotent operations (i.e. operation that can be executed multiple times without side effects) and using a simple timeout mechanism. Figure 7.14 shows the state machines for "at least once" reliability using implicit acknowledgements. When the client sends out the request it starts a timer with the timeout specified for the specific method. If no response is received before the timer expires (round transition at the top), the client will retry the operation. A Typical number of retries would be 2, so that 3 requests are sent.

The number of retries, the timeout values, and the timeout behavior (constant or exponential back off) are outside of the SOME/IP specification.

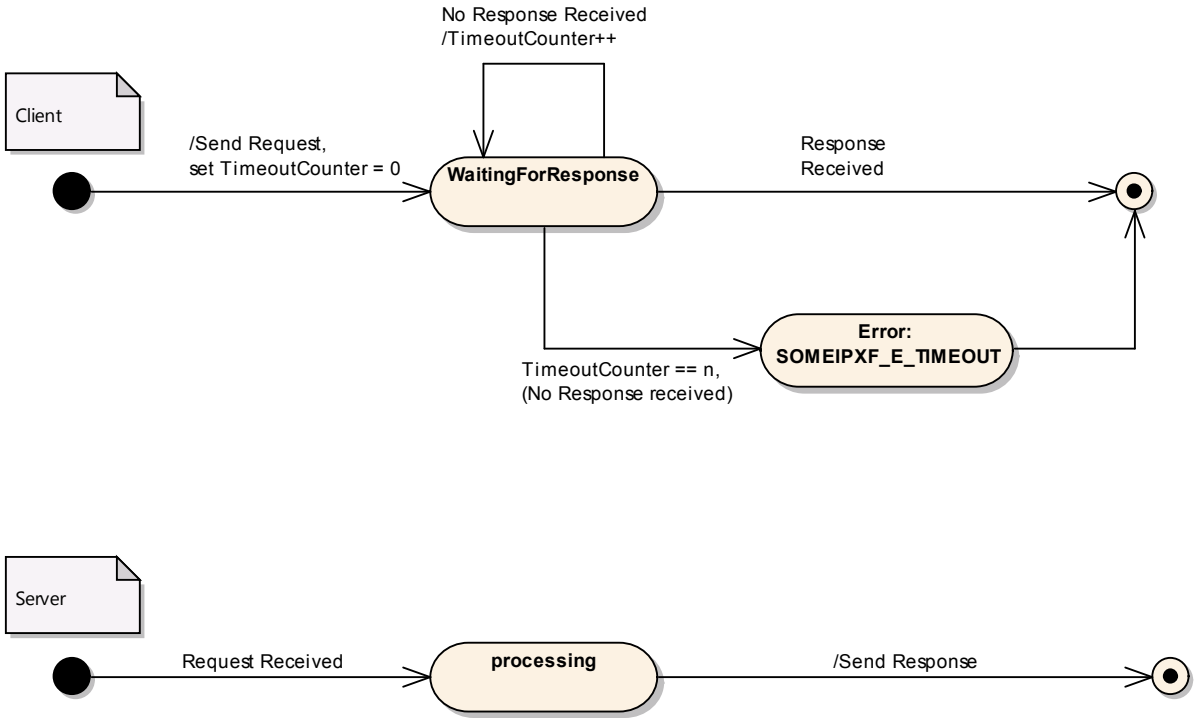


**Figure 7.14: State Machines for Reliability "At least once" (idempotent operations)**

### 7.2.5 Security Events

#### 7.2.5.1 Security Event Overview

**[SWS_SomeIpXf_00116] Security Event Reporting** ⌈If security event reporting has been enabled for the SOME/IP transformer module (XfrmEnableSecurityEventReporting = true) the respective security events shall be reported to the IdsM via the interfaces defined in [14, SWS BSW General].⌋

**[SWS_SomeIpXf_00308] Security events for SomeIpXf** ⌈

| *Name* | *Description* | *ID* |
|---|---|---|
| SEV_SOMEIP_ERROR | The processing of a SOME/IP message resulted in a return not equal to E_OK. | 138 |

⌋

**[SWS_SomeIpXf_00117] Security event trigger SEV_SOMEIP_ERROR** ⌈If a PDU is received, which generates one of the following return codes, SOMEIP Transformer shall raise the security event SEV_SOMEIP_ERROR: E_NOT_OK, E_WRONG_PROTOCOL_VERSION, E_WRONG_INTERFACE_VERSION, E_MALFORMED_MESSAGE, E_WRONG_MESSAGE_TYPE.⌋

**[SWS_SomeIpXf_00309] Security event context data definition: SEV_SOMEIP_ERROR** ⌈

| *SEV Name* | *SEV_SOMEIP_ERROR* | |
|---|---|---|
| *ID* | 138 | |
| *Description* | The processing of a SOME/IP message resulted in a return not equal to E_OK. | |
| *Context Data Version* | 1 | |
| *Context Data* | *Data Type* | *Allowed Values* |
| SourceIpAddress | uint8 [16] | All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2 |
| SourcePort | uint16 | |
| DestinationIpAddress | uint8 [16] | All IPv6 addresses and IPv4 addresses shall be encoded as specified in RFC 4291 Section 2.5.5.2 |
| DestinationPort | uint16 | |
| SomeIpHeader | uint8 [16] | |
| ReturnCode | uint8 | E_NOT_OK (0x01)<br>E_UNKNOWN_SERVICE (0x02)<br>E_UNKNOWN_METHOD (0x03)<br>E_WRONG_PROTOCOL_VERSION (0x07)<br>E_WRONG_INTERFACE_VERSION (0x08)<br>E_MALFORMED_MESSAGE (0x09)<br>E_WRONG_MESSAGE_TYPE (0x0A)<br>E_E2E (0x0D) |

⌋

## 7.3 Error Classification

Chapter [14, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of following error types which may occur in [14, SWS BSW General] modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

### 7.3.1 Development Errors

**[SWS_SomeIPxf_00184] Definition of development errors in module SomeIpXf**

*Upstream requirements:* SRS_BSW_00337

⌈

| *Type of error* | *Related error code* | *Error value* |
|---|---|---|
| Error code if any API service is called before the transformer module was initialized with Init or after a call to DeInit | SOMEIPXF_E_UNINIT | 0x01 |
| Error code if an invalid configuration set was selected | SOMEIPXF_E_INIT_FAILED | 0x02 |

▽

△

| *Type of error* | *Related error code* | *Error value* |
|---|---|---|
| API service called with wrong parameter | SOMEIPXF_E_PARAM | 0x03 |
| API service called with invalid pointer | SOMEIPXF_E_PARAM_POINTER | 0x04 |

⌋

### 7.3.2 Runtime Errors

There are no runtime errors.

### 7.3.3 Production Errors

There are no production errors.

# 8 API specification

## 8.1 Imported types

There are no imported types from other modules beyond those specified in [3, SWS Transformer General].

In the Module Interlink Headers file which is imported by the SOME/IP Transformer, all `ImplementationDataType`s known to the RTE are included. Using this mechanism, the SOME/IP Transformer knows all data types of data which shall be transformed.

**[SWS_SomeIpXf_91002] Definition of imported datatypes of module SomeIpXf**

*Upstream requirements:* SRS_Xfrm_00002

⌈

| *Module* | *Header File* | *Imported Type* |
|---|---|---|
| Dem | Rte_Dem_Type.h | Dem_EventIdType |
| | Rte_Dem_Type.h | Dem_EventStatusType |
| Rte | SchM_<Mip>.h | Rte_Cs_TransactionHandleType |
| Std | Std_Types.h | Std_MessageResultType |
| | Std_Types.h | Std_MessageTypeType |
| | Std_Types.h | Std_ReturnType |

⌋

## 8.2 Type definitions

**[SWS_SomeIpXf_00183] Definition of datatype SomeIpXf_ConfigType**

*Upstream requirements:* SRS_BSW_00404, SRS_BSW_00441, SRS_BSW_00389, SRS_BSW_00388

⌈

| | | |
|---|---|---|
| ***Name*** | SomeIpXf_ConfigType | |
| ***Kind*** | Structure | |
| ***Elements*** | implementation specific | |
| | ***Type*** | – |
| | ***Comment*** | – |
| ***Description*** | This is the type of the data structure containing the initialization data for the transformer. | |
| ***Available via*** | SomeIpXf.h | |

⌋

## 8.3 Function definitions

This chapter contains all APIs provided by the module

The SOME/IP transformer provides the specific interfaces generally required by [3, SWS Transformer General].

**[SWS_SomeIpXf_00150]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The SOME/IP Transformer shall only provide functions for transformers where the `TransformationTechnology` is referenced as the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology`.⌋

That means, only the first transformer in a transformer chain can be a SOME/IP Transformer because serializer transformer are in general only allowed to be the first transformer in a chain.

### 8.3.1 SomeIpXf_ExtractProtocolHeaderFields

**[SWS_SomeIpXf_91001] Definition of API function SomeIpXf_ExtractProtocol HeaderFields**

*Upstream requirements:* SRS_Xfrm_00002

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_ExtractProtocolHeaderFields | |
| ***Syntax*** | `Std_ReturnType SomeIpXf_ExtractProtocolHeaderFields ( const uint8* buffer, uint32 bufferLength, Std_MessageTypeType* messageType, Std_MessageResultType* messageResult )` | |
| ***Service ID [hex]*** | 0x05 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | buffer | Buffer allocated by the RTE, where the transformed data has to be stored by the transformer. |
| | bufferLength | Length of the buffer |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | messageType | Canonical representation of the message type (extracted from the transformers protocol header). |
| | messageResult | Canonical representation of the message result type (extracted from the transformers protocol header). |
| ***Return value*** | Std_ReturnType | `E_OK`: Relevant protocol header fields have been extracted successfully.<br>`E_NOT_OK`: An error occurred during parsing of the SOME/IP protocol header (e.g., incorrect protocol version or insufficient buffer length (bufferLength smaller than minimal SOME/IPheader length)) |

▽

△

| | |
|---|---|
| ***Description*** | Function to extract the relevant SOME/IP protocol header fields of the message and the type of the message result. - At the time being, this is limited to the types used for C/S communication (i.e., REQUEST and RESPONSE and OK and ERROR). |
| ***Available via*** | SomeIpXf.h |

⌋

**[SWS_SomeIpXf_00296]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall extract the type of a message and the type of the message result from the SOME/IP protocol header and provide this information in a canonical representation via its output arguments.⌋

**[SWS_SomeIpXf_00297]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall check whether the provided `bufferLength` is larger or equal than the size of the protocol header processed by the SOME/IP transformer (i.e., 8 bytes). – If this is not the case, `E_NOT_OK` shall be returned. Neither `messageType` nor `messageResult` shall be modified in this case.⌋

**[SWS_SomeIpXf_00298]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall check whether the value of the Protocol Version field (see [PRS_SOMEIP_00052]) is equal to the value defined by [PRS_SOMEIP_00051]. – If this is not the case, `E_NOT_OK` shall be returned. Neither `messageType` nor `messageResult` shall be modified in this case.⌋

**[SWS_SomeIpXf_00299]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall check whether the value of the Message Type field (see [PRS_SOMEIP_00055]) is equal `REQUEST`, `RESPONSE`, or `ERROR`. – If this is not the case, `E_NOT_OK` shall be returned. Neither `messageType` nor `messageResult` shall be modified in this case.⌋

**[SWS_SomeIpXf_00301]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall return `E_OK` in all other cases.⌋

**[SWS_SomeIpXf_00302]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall set `messageType` to `STD_MESSAGETYPE_REQUEST` in case the value of the Message Type field (see [PRS_SOMEIP_00055]) is equal `REQUEST`.⌋

**[SWS_SomeIpXf_00303]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall set `messageType` to `STD_MESSAGETYPE_RESPONSE` in case the value of the Message Type field (see [PRS_SOMEIP_00055]) is equal `RESULT` or `ERROR`.⌋

**[SWS_SomeIpXf_00304]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall set `messageResult` to `STD_MESSAGERESULT_ERROR` in case the value of the Message Type field (see [PRS_SOMEIP_00055]) is equal to `ERROR` or if the value of the Return Code field (see [PRS_SOMEIP_00058]) is different from 0.⌋

**[SWS_SomeIpXf_00305]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_ExtractProtocolHeaderFields` specified in [SWS_SomeIpXf_91001] shall set `messageResult` to `STD_MESSAGERESULT_OK` otherwise (i.e., in case the value of the Message Type field (see [PRS_SOMEIP_00055]) is different from `ERROR` and if the value of the Return Code field (see [PRS_SOMEIP_00058]) is 0.⌋

### 8.3.2 SomeIpXf_<transformerId>

**[SWS_SomeIpXf_00138] Definition of API function SomeIpXf_<transformerId>**

*Upstream requirements:* SRS_Xfrm_00101, SRS_Xfrm_00009, SRS_BSW_00494, SRS_BSW_00486, SRS_BSW_00485, SRS_BSW_00484, SRS_BSW_00462, SRS_BSW_00432, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00392, SRS_BSW_00369, SRS_BSW_00357

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_<transformerId> | |
| ***Syntax*** | `uint8 SomeIpXf_<transformerId> (`<br>`  uint8* buffer,`<br>`  uint32* bufferLength,`<br>`  <paramtype> dataElement`<br>`)` | |
| ***Service ID [hex]*** | 0x03 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | dataElement | Data element which shall be transformed |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | buffer | Buffer allocated by the RTE, where the transformed data has to be stored by the transformer |
| | bufferLength | Used length of the buffer |
| ***Return value*** | uint8 | 0x00 (`E_OK`): Serialization successful<br>0x81 (`E_SER_GENERIC_ERROR`): A generic error occurred |
| ***Description*** | This function transforms a Sender/Receiver communication using the serialization of SOME/IP. It takes the data element as input and outputs a uint8 array containing the serialized data.<br>The length of the serialized data shall be calculated by the transformer during runtime and returned in the OUT-parameter bufferLength. It may be smaller than the maximum buffer size used by the RTE for buffer allocation. | |
| ***Available via*** | SomeIpXf.h | |

⌋

**[SWS_SomeIpXf_00228]**

*Upstream requirements:* SRS_Xfrm_00102

⌈In function `SomeIpXf_<transformerId>` defined in [SWS_SomeIpXf_00138]

- `paramtype` is derived from `type` according to the parameter passing rules rules defined by the [6, SRS BSW General] (see [SRS_BSW_00484], [SRS_BSW_00485], and [SRS_BSW_00486]) and [14, SWS BSW General] (see [SWS_BSW_00186]).
- `type` shall be the data type of the data element after all data conversion activities of the RTE
- `transformerId` shall be the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, SWS Transformer General])

⌋

This function specified in [SWS_SomeIpXf_00138] exists for each transformed Sender/Receiver communication which uses the SOME/IP serialization.

**[SWS_SomeIpXf_00139]**

*Upstream requirements:* SRS_Xfrm_00102

⌈The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00138] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `SenderReceiverToSignalMapping`.⌋

**[SWS_SomeIpXf_00140]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00138] shall serialize primitive or complex data elements of Sender/Receiver communication into a linear byte array representation using the SOME/IP serialization.⌋

**[SWS_SomeIpXf_00214]**

*Upstream requirements:* SRS_Xfrm_00002

⌈After serialization of the data, the function `SomeIpXf_<transformerId>` specified in [[SWS_SomeIpXf_00138] shall increment the Session Handling ID counter assigned to `<transformerId>` if transmission path includes SomeIpTp.⌋

**[SWS_SomeIpXf_00215]**

*Upstream requirements:* SRS_Xfrm_00101

⌈When the Session Handling ID counter assigned to `<transformerId>` is `0xFFFF` and gets incremented, it shall roll-over to `0x0001` (instead of `0x0000`) if transmission path includes SomeIpTp.⌋

**[SWS_SomeIpXf_00141] Definition of API function SomeIpXf_<transformerId>**

*Upstream requirements:* SRS_Xfrm_00101

⌈

| ***Service Name*** | SomeIpXf_<transformerId> |
|---|---|
| ***Syntax*** | `uint8 SomeIpXf_<transformerId> (`<br>`  const Rte_Cs_TransactionHandleType* TransactionHandle,`<br>`  uint8* buffer,`<br>`  uint32* bufferLength,`<br>`  [Std_ReturnType returnValue],`<br>`  <paramtype> data_1, ...`<br>`  <paramtype> data_n`<br>`)` |
| ***Service ID [hex]*** | 0x03 |
| ***Sync/Async*** | Synchronous |
| ***Reentrancy*** | Non Reentrant |

▽

△

| | | |
|---|---|---|
| ***Parameters (in)*** | TransactionHandle | Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests. |
| | returnValue | Return value from server side for transmission to the calling client. This argument is only available for serializers of the response of a Client/Server communication if<br>• the ClientServerOperation has at least one PossibleError defined or<br>• autonomous error reaction is activated |
| | data_1 | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| | ... | ... |
| | data_n | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | buffer | Buffer allocated by the RTE, where the transformed data has to be stored by the transformer |
| | bufferLength | Used length of the buffer |
| ***Return value*** | uint8 | 0x00 (`E_OK`): Serialization successful<br>0x81 (`E_SER_GENERIC_ERROR`): A generic error occurred |
| ***Description*** | This function transforms a Client/Server communication using the serialization of SOME/IP. It takes the operation arguments and optionally the return value as input and outputs a uint8 array containing the serialized data.<br>The length of the serialized data shall be calculated by the transformer during runtime and returned in the OUT-parameter bufferLength. It may be smaller than the maximum buffer size used by the RTE for buffer allocation. | |
| ***Available via*** | SomeIpXf.h | |

⌋

**[SWS_SomeIpXf_00229]**

*Upstream requirements:* SRS_Xfrm_00101

⌈In function `SomeIpXf_<transformerId>` defined in [SWS_SomeIpXf_00141]

- `paramtype` is derived from `type` according to the parameter passing rules rules defined by the [6, SRS BSW General] (see [SRS_BSW_00484], [SRS_BSW_00485], and [SRS_BSW_00486]) and [14, SWS BSW General] (see [SWS_BSW_00186]).
- `type` shall be the data type of the data element after all data conversion activities of the RTE
- `transformerId` shall be the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, SWS Transformer General]).

⌋

This function specified in [SWS_SomeIpXf_00141] exists for the server and each client of each transformed Client/Server communication which uses the SOME/IP serialization.

It exists on both the Client and the Server but the arguments are different.

On the client it serializes the request of the Client/Server call. There, the `data_1`, ..., `data_n` arguments of the API correspond to the *IN* and *INOUT* arguments of the `ClientServerOperation`. The argument `returnValue` doesn't exist.

On the server it serializes the response of the Client/Server call. There, the `data_1`, ..., `data_n` arguments of the API correspond to the *INOUT* and *OUT* arguments of the `ClientServerOperation`. The argument `returnValue` exists here if at least one `possibleError` is defined for the `ClientServerOperation` because the return code of the operation has to be transmitted.

**[SWS_SomeIpXf_00142]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00141] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `ClientServerToSignalMapping` in the `callSignal` or `returnSignal`.⌋

Due to [SWS_SomeIpXf_00142], the API of [SWS_SomeIpXf_00141] exists both on client and server.

**[SWS_SomeIpXf_00143]**

*Upstream requirements:* SRS_Xfrm_00101

⌈The function `SomeIpXf_<transformerId>[_<symbolSuffix>]` specified in [SWS_SomeIpXf_00141] shall serialize all primitive or complex operation arguments and the return value (if executed on server side) of Client/Server communication into a linear byte array representation using the SOME/IP serialization.⌋

**[SWS_SomeIpXf_00203]**

*Upstream requirements:* SRS_Xfrm_00105

⌈The function `SomeIpXf_<transformerId>[_<symbolSuffix>]` specified in [SWS_SomeIpXf_00141] shall ignore all arguments `data_1, ..., data_n` if the return code is greater or equal to `0x80` because they are not filled with meaningful values.⌋

### [SWS_SomeIpXf_00206] Definition of API function SomeIpXf_<transformerId>

*Upstream requirements:* SRS_Xfrm_00002

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_<transformerId> | |
| ***Syntax*** | `uint8 SomeIpXf_<transformerId> ( uint8* buffer, uint32* bufferLength )` | |
| ***Service ID [hex]*** | 0x03 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | None | |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | buffer | Buffer allocated by the RTE, where the transformed data has to be stored by the transformer |
| | bufferLength | Used length of the buffer |
| ***Return value*** | uint8 | 0x00 (`E_OK`): Serialization successful<br>0x81 (`E_SER_GENERIC_ERROR`): A generic error occurred |
| ***Description*** | This function transforms an external trigger event using the serialization of SOME/IP. It takes trigger as input and outputs a uint8 array.<br>The length of the transformed data shall be calculated by the transformer during runtime and returned in the OUT parameter bufferLength. It may be smaller than the maximum buffer size used by the RTE for buffer allocation. | |
| ***Available via*** | SomeIpXf.h | |

⌋

### [SWS_SomeIpXf_00230]

*Upstream requirements:* SRS_Xfrm_00101

⌈In function `SomeIpXf_<transformerId>` defined in [SWS_SomeIpXf_00206]

- `transformerId` shall be the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, SWS Transformer General]).

⌋

This function specified in [SWS_SomeIpXf_00206] exists on the trigger source side for each transformed external trigger event which uses SOME/IP transformation.

### [SWS_SomeIpXf_00207]

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00206] shall exist for the first referenced `TransformationTechnology` in the ordered `transformerChain` of a `DataTransformation` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by a `TriggerToSignalMapping`.⌋

**[SWS_SomeIpXf_00208]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00206] shall serialize an external trigger event into a linear byte array representation using the SOME/IP serialization.⌋

As an external trigger event consists of an `ISignal` with length equal to zero, the serialized SOME/IP message only contains a header but no payload.

### 8.3.3 SomeIpXf_Inv_<transformerId>

**[SWS_SomeIpXf_00144] Definition of API function SomeIpXf_Inv_<transformerId>**

*Upstream requirements:* SRS_Xfrm_00009, SRS_Xfrm_00007, SRS_BSW_00494, SRS_BSW_00485, SRS_BSW_00484, SRS_BSW_00462, SRS_BSW_00432, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00392, SRS_BSW_00383, SRS_BSW_00369, SRS_BSW_00357

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_Inv_<transformerId> | |
| ***Syntax*** | `uint8 SomeIpXf_Inv_<transformerId> ( const uint8* buffer, uint32 bufferLength, <type>* dataElement )` | |
| ***Service ID [hex]*** | 0x04 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | buffer | Buffer allocated by the RTE, where the still serialized data are stored by the Rte |
| | bufferLength | Used length of the buffer |
| ***Parameters (inout)*** | dataElement | Data element which is the result of the transformation and contains the deserialized data element |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint8 | 0x00 (`E_OK`): Deserialization successful<br>0x01 (`E_NO_DATA`): No data available which can be deserialized<br>0x81 (`E_SER_GENERIC_ERROR`): A generic error occurred<br>0x87 (`E_SER_WRONG_PROTOCOL_VERSION`): The version of the receiving transformer didn't match the sending transformer.<br>0x88 (`E_SER_WRONG_INTERFACE_VERSION`): Interface version of serialized data is not supported.<br>0x89 (`E_SER_MALFORMED_MESSAGE`): The received message is malformed. The transformer is not able to produce an output.<br>0x8a (`E_SER_WRONG_MESSAGE_TYPE`): The received message type was not expected. |
| ***Description*** | This function deserializes a Sender/Receiver communication using the deserialization of SOME/IP. It takes the uint8 array containing the serialized data as input and outputs the original data element which will be passed to the RTE. | |
| ***Available via*** | SomeIpXf.h | |

⌋

**Note:** If variable size arrays with arrayImplPolicy set to payloadAsPointerToArray are received as serialized data input, the transformer may need to update the outgoing dataElement in response to the size and location of the payload once deserialised.

**[SWS_SomeIpXf_00231]**

*Upstream requirements:* SRS_Xfrm_00101

⌈In function `SomeIpXf_Inv_<transformerId>` defined in [SWS_SomeIpXf_00144]

- `type` shall be the data type of the data element before all data conversion activities of the RTE
- `transformerId` shall be the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, SWS Transformer General]).

⌋

This function specified in [SWS_SomeIpXf_00144] exists for each transformed Sender/Receiver communication which uses the SOME/IP serialization.

**[SWS_SomeIpXf_00146]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00144] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `SenderReceiverToSignalMapping`.⌋

**[SWS_SomeIpXf_00147]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00144] shall deserialize a linear byte array to primitive or complex data elements of Sender/Receiver communication using the SOME/IP deserialization.⌋

**[SWS_SomeIpXf_00264]**

*Upstream requirements:* SRS_Xfrm_00001, SRS_Xfrm_00004

⌈If `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00144] is called with buffer equal to `NULL_PTR` and bufferLength equal to 0, the output buffer *buffer* shall not be changed and `SomeIpXf_Inv_<transformerId>` shall return with `E_NO_DATA`.⌋

### [SWS_SomeIpXf_00145] Definition of API function SomeIpXf_Inv_<transformer Id>

*Upstream requirements:* SRS_Xfrm_00106

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_Inv_<transformerId> | |
| ***Syntax*** | `uint8 SomeIpXf_Inv_<transformerId> (`<br>`  Rte_Cs_TransactionHandleType* TransactionHandle,`<br>`  const uint8* buffer,`<br>`  uint32 bufferLength,`<br>`  [Std_ReturnType* returnValue],`<br>`  [<type>* data_1, ...`<br>`  <type>* data_n]`<br>`)` | |
| ***Service ID [hex]*** | 0x04 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | buffer | Buffer allocated by the RTE, where the still serialized data are stored by the Rte |
| | bufferLength | Used length of the buffer |
| ***Parameters (inout)*** | data_1 | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| | ... | ... |
| | data_n | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| ***Parameters (out)*** | TransactionHandle | Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests. |
| | returnValue | Return value from server side for transmission to the calling client. This argument is only available for serializers of the response of a Client/Server communication if<br>• the ClientServerOperation has at least one PossibleError defined or<br>• autonomous error reaction is activated |
| ***Return value*** | uint8 | 0x00 (`E_OK`): Deserialization successful<br>0x01 (`E_NO_DATA`): No data available which can be deserialized<br>0x81 (`E_SER_GENERIC_ERROR`): A generic error occurred<br>0x87 (`E_SER_WRONG_PROTOCOL_VERSION`): The version of the receiving transformer didn't match the sending transformer.<br>0x88 (`E_SER_WRONG_INTERFACE_VERSION`): Interface version of serialized data is not supported.<br>0x89 (`E_SER_MALFORMED_MESSAGE`): The received message is malformed. The transformer is not able to produce an output.<br>0x8a (`E_SER_WRONG_MESSAGE_TYPE`): The received message type was not expected. |
| ***Description*** | This function deserializes a Client/Server communication using the deserialization of SOME/IP. It takes the uint8 array containing the serialized data as input and outputs the return value of the server runnable and the operation arguments which have to be passed from the server to the client. | |
| ***Available via*** | SomeIpXf.h | |

⌋

**Note:** If variable size arrays with arrayImplPolicy set to payloadAsPointerToArray are received as serialized data input, the transformer may need to update the outgoing parameters `data_1`, ..., `data_n` in response to the size and location of the payload once deserialised.

**[SWS_SomeIpXf_00232]**

*Upstream requirements:* SRS_Xfrm_00101

⌈In function `SomeIpXf_Inv_<transformerId>` defined in [SWS_SomeIpXf_00145]

- `paramtype` is derived from `type` according to the parameter passing rules rules defined by the [6, SRS BSW General] (see [SRS_BSW_00484], [SRS_BSW_00485], and [SRS_BSW_00486]) and [14, SWS BSW General] (see [SWS_BSW_00186]).
- `type` shall be the data type of the data element before all data conversion activities of the RTE
- `transformerId` shall be the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, SWS Transformer General]).

⌋

This function specified in [SWS_SomeIpXf_00145] exists for the server and each client of each transformed Client/Server communication which uses the SOME/IP serialization.

It exists on both the Client and the Server but the arguments are different.

On the server it deserializes the request of the Client/Server call. There, the `data_1`, ..., `data_n` arguments of the API correspond to the *IN* and *INOUT* arguments of the `ClientServerOperation`. The argument `returnValue` doesn't exist.

On the client it deserializes the response of the Client/Server call. There, the `data_1`, ..., `data_n` arguments of the API correspond to the *INOUT* and *OUT* arguments of the `ClientServerOperation`. If the ClientServerOperation has at least one possibleError defined, the returnValue shall be determined by subtracting 0x1F from the Return Code value. Otherwise the return value shall be set to the actual value of the Return Code.

**[SWS_SomeIpXf_00148]**

*Upstream requirements:* SRS_Xfrm_00101

⌈

The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00145] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `ClientServerToSignalMapping` in the `callSignal` or `returnSignal`.⌋

Due to [SWS_SomeIpXf_00148], the API of [SWS_SomeIpXf_00145] exists both on client and server.

**[SWS_SomeIpXf_00149]**

*Upstream requirements:* SRS_Xfrm_00106

⌈The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00145] shall deserialize a linear byte array which contains primitive or complex operation arguments and the return value (if executed on client side) of Client/Server communication using the SOME/IP deserialization. If MessageType is ERROR(0x81) the payload of the message shall not be deserialized, but the returnCode shall be sett according to [SWS_SomeIpXf_00232].⌋

**[SWS_SomeIpXf_00265]**

*Upstream requirements:* SRS_Xfrm_00001, SRS_Xfrm_00004

⌈If `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00145] is called with buffer equal to `NULL_PTR` and bufferLength equal to 0, the output buffer *buffer* shall not be changed and `SomeIpXf_Inv_<transformerId>` shall return with `E_NO_DATA`.⌋

**[SWS_SomeIpXf_00209] Definition of API function SomeIpXf_Inv_<transformer Id>**

*Upstream requirements:* SRS_Xfrm_00002

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_Inv_<transformerId> | |
| ***Syntax*** | `uint8 SomeIpXf_Inv_<transformerId> ( const uint8* buffer, uint32 bufferLength )` | |
| ***Service ID [hex]*** | 0x04 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | buffer | Buffer allocated by the RTE, where the still serialized data are stored by the Rte |
| | bufferLength | Used length of the buffer |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | uint8 | 0x00 (`E_OK`): Deserialization successful<br>0x01 (`E_NO_DATA`): No data available which can be deserialized<br>0x81 (`E_SER_GENERIC_ERROR`): A generic error occurred<br>0x87 (`E_SER_WRONG_PROTOCOL_VERSION`): The version of the receiving transformer didn't match the sending transformer.<br>0x88 (`E_SER_WRONG_INTERFACE_VERSION`): Interface version of serialized data is not supported.<br>0x89 (`E_SER_MALFORMED_MESSAGE`): deprecated.<br>0x8a (`E_SER_WRONG_MESSAGE_TYPE`): The received message type was not expected. |
| ***Description*** | This function deserializes an external trigger event using the deserialization of SOME/IP. | |
| ***Available via*** | SomeIpXf.h | |

⌋

**[SWS_SomeIpXf_00233]**

*Upstream requirements:* SRS_Xfrm_00101

⌈In function `SomeIpXf_Inv_<transformerId>` defined in [SWS_SomeIpXf_00209]

- `transformerId` shall be the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, SWS Transformer General]).

⌋

This function specified in [SWS_SomeIpXf_00209] exists on the trigger sink side for each transformed external trigger event which uses SOME/IP transformation.

**[SWS_SomeIpXf_00210]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00209] shall exist for the first referenced `TransformationTechnology` in the ordered `transformerChain` of a `DataTransformation` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by a `TriggerToSignalMapping`.⌋

**[SWS_SomeIpXf_00211]**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00209] shall deserialize a linear byte array to an external trigger event using the SOME/IP deserialization.⌋

**[SWS_SomeIpXf_00266]**

*Upstream requirements:* SRS_Xfrm_00001, SRS_Xfrm_00004

⌈If `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00209] is called with buffer equal to `NULL_PTR` and bufferLength equal to 0,`SomeIpXf_Inv_<transformerId>` shall return with `E_NO_DATA`.⌋

As an external trigger event consists of an `ISignal` with length = 64 Bit, the serialized SOME/IP message only contains a header but no payload.

### 8.3.4 SomeIpXf_Init

**[SWS_SomeIpXf_00181] Definition of API function SomeIpXf_Init**

*Upstream requirements:* SRS_BSW_00411, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00459, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00478, SRS_BSW_00479, SRS_BSW_00480, SRS_BSW_00483, SRS_BSW_00484, SRS_BSW_00485, SRS_BSW_00486, SRS_BSW_00494, SRS_Xfrm_00201, SRS_Xfrm_00011, SRS_Xfrm_00010, SRS_Xfrm_00006, SRS_BSW_00448, SRS_BSW_00432, SRS_BSW_00419, SRS_BSW_00413, SRS_BSW_00403, SRS_BSW_00401, SRS_BSW_00399, SRS_BSW_00398, SRS_BSW_00396, SRS_BSW_00395, SRS_BSW_00390, SRS_BSW_00358, SRS_BSW_00351, SRS_BSW_00350, SRS_BSW_00162, SRS_BSW_00161, SRS_BSW_00005

⌈

| | | |
|---|---|---|
| ***Service Name*** | SomeIpXf_Init | |
| ***Syntax*** | `void SomeIpXf_Init (`<br>`  const SomeIpXf_ConfigType* config`<br>`)` | |
| ***Service ID [hex]*** | 0x01 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | config | Pointer to the transformer's configuration data. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | This service initializes the transformer for the further processing. | |
| ***Available via*** | SomeIpXf.h | |

⌋

### 8.3.5 SomeIpXf_DeInit

**[SWS_SomeIpXf_00182] Definition of API function SomeIpXf_DeInit**

*Upstream requirements:* SRS_BSW_00411, SRS_BSW_00336, SRS_BSW_00345

⌈

| | |
|---|---|
| ***Service Name*** | SomeIpXf_DeInit |
| ***Syntax*** | `void SomeIpXf_DeInit (`<br>`  void`<br>`)` |
| ***Service ID [hex]*** | 0x02 |
| ***Sync/Async*** | Synchronous |
| ***Reentrancy*** | Non Reentrant |
| ***Parameters (in)*** | None |
| ***Parameters (inout)*** | None |
| ***Parameters (out)*** | None |

▽

△

| | |
|---|---|
| ***Return value*** | None |
| ***Description*** | This service deinitializes the transformer. |
| ***Available via*** | SomeIpXf.h |

⌋

## 8.4 Scheduled functions

SOME/IP Transformer has no scheduled functions.

## 8.5 Expected interfaces

In this chapter, all interfaces required from other modules are listed.

### 8.5.1 Mandatory Interfaces

There are no mandatory interfaces, which are required to fulfill the core functionality of the module.

### 8.5.2 Optional Interfaces

This chapter defines all interfaces, which are required to fulfill an optional functionality of the module.

**[SWS_SomeIpXf_91003] Definition of optional interfaces requested by module SomeIpXf**

*Upstream requirements:* SRS_BSW_00170, SRS_BSW_00350

⌈

| ***API Function*** | ***Header File*** | ***Description*** |
|---|---|---|
| Dem_SetEventStatus | Dem.h | Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING) |

⌋

# 9 Sequence diagrams

There are no sequence diagrams applicable to SOME/IP Transformer.

# 10 Configuration specification

There is no module specific configuration available to the SOME/IP Transformer. The EcuC defined in [3, SWS Transformer General] shall be used.

**[SWS_SomeIpXf_00185]**

*Upstream requirements:* SRS_BSW_00159

⌈The `apiServicePrefix` of the SOME/IP transformer's EcuC shall be set to `SomeIpXf`.⌋

# A Not applicable requirements

**[SWS_SomeIpXf_00100] Not applicable requirements**

*Upstream requirements:* SRS_BSW_00384, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00375, SRS_BSW_00168, SRS_BSW_00416, SRS_BSW_00393, SRS_BSW_00004

⌈These requirements are not applicable to this specification.⌋

# B Change History

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

## B.1 Change History R26-11

### B.1.1 Added Specification Items in R26-11

[SWS_SomeIpXf_00100] [SWS_SomeIpXf_00116] [SWS_SomeIpXf_00117] [SWS_SomeIpXf_00308] [SWS_SomeIpXf_00321] [SWS_SomeIpXf_00322] [SWS_SomeIpXf_00323] [SWS_SomeIpXf_00324] [SWS_SomeIpXf_00325] [SWS_SomeIpXf_00326]

### B.1.2 Changed Specification Items in R26-11

[SWS_SomeIPxf_00184] [SWS_SomeIpXf_00060] [SWS_SomeIpXf_00244] [SWS_SomeIpXf_00246] [SWS_SomeIpXf_00309] [SWS_SomeIpXf_91002]

### B.1.3 Deleted Specification Items in R26-11

[SWS_SomeIpXf_00180]

### B.1.4 Added Constraints in R26-11

none

### B.1.5 Changed Constraints in R26-11

none

### B.1.6 Deleted Constraints in R26-11

none

## B.2 Change History R25-11

### B.2.1 Added Specification Items in R25-11

[SWS_SomeIpXf_00312] [SWS_SomeIpXf_00313] [SWS_SomeIpXf_00314] [SWS_SomeIpXf_00315] [SWS_SomeIpXf_00316] [SWS_SomeIpXf_00317] [SWS_SomeIpXf_00318] [SWS_SomeIpXf_00319] [SWS_SomeIpXf_00320] [SWS_SomeIpXf_91003]

### B.2.2 Changed Specification Items in R25-11

[SWS_SomeIpXf_00017] [SWS_SomeIpXf_00115] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00145] [SWS_SomeIpXf_00172] [SWS_SomeIpXf_00209] [SWS_SomeIpXf_00223] [SWS_SomeIpXf_00234] [SWS_SomeIpXf_00244] [SWS_SomeIpXf_00245] [SWS_SomeIpXf_00258] [SWS_SomeIpXf_00266] [SWS_SomeIpXf_00271] [SWS_SomeIpXf_91002]

### B.2.3 Deleted Specification Items in R25-11

[SWS_SomeIpXf_00239]

### B.2.4 Added Constraints in R25-11

none

### B.2.5 Changed Constraints in R25-11

none

### B.2.6 Deleted Constraints in R25-11

none

## B.3 Change History R24-11

### B.3.1 Added Specification Items in R24-11

[SWS_SomeIpXf_00310] [SWS_SomeIpXf_00311] [SWS_SomeIpXf_91002]

### B.3.2 Changed Specification Items in R24-11

[SWS_SomeIpXf_00017] [SWS_SomeIpXf_00054] [SWS_SomeIpXf_00111] [SWS_SomeIpXf_00149] [SWS_SomeIpXf_00181] [SWS_SomeIpXf_00182] [SWS_SomeIpXf_00201] [SWS_SomeIpXf_00242] [SWS_SomeIpXf_00245]

### B.3.3 Deleted Specification Items in R24-11

none

### B.3.4 Added Constraints in R24-11

[CP_SWS_SomeIpXf_CONSTR_00001] [CP_SWS_SomeIpXf_CONSTR_00002]

### B.3.5 Changed Constraints in R24-11

none

### B.3.6 Deleted Constraints in R24-11

[SWS_SomeIpXf_CONSTR_00001] [SWS_SomeIpXf_CONSTR_00002]

## B.4 Change History R23-11

### B.4.1 Added Specification Items in R23-11

[SWS_SomeIpXf_00309]

### B.4.2 Changed Specification Items in R23-11

[SWS_SomeIpXf_00024] [SWS_SomeIpXf_00031] [SWS_SomeIpXf_00036] [SWS_SomeIpXf_00088] [SWS_SomeIpXf_00108] [SWS_SomeIpXf_00115] [SWS_SomeIpXf_00168] [SWS_SomeIpXf_00172] [SWS_SomeIpXf_00183] [SWS_SomeIpXf_00204] [SWS_SomeIpXf_00212] [SWS_SomeIpXf_00214] [SWS_SomeIpXf_00215] [SWS_SomeIpXf_00270]

### B.4.3 Deleted Specification Items in R23-11

[SWS_SomeIpXf_00013] [SWS_SomeIpXf_00136]

### B.4.4 Added Constraints in R23-11

[SWS_SomeIpXf_CONSTR_00001] [SWS_SomeIpXf_CONSTR_00002]

### B.4.5 Changed Constraints in R23-11

none

### B.4.6 Deleted Constraints in R23-11

[SWS_SomeIpXf_CONSTR_0001] [SWS_SomeIpXf_CONSTR_0002]

## B.5 Change History R22-11

### B.5.1 Added Specification Items in R22-11

none

### B.5.2 Changed Specification Items in R22-11

[SWS_SomeIPxf_00184] [SWS_SomeIpXf_00054] [SWS_SomeIpXf_00138] [SWS_SomeIpXf_00141] [SWS_SomeIpXf_00144] [SWS_SomeIpXf_00145] [SWS_SomeIpXf_00152] [SWS_SomeIpXf_00180] [SWS_SomeIpXf_00181] [SWS_SomeIpXf_00182] [SWS_SomeIpXf_00183] [SWS_SomeIpXf_00200] [SWS_SomeIpXf_00206] [SWS_SomeIpXf_00209] [SWS_SomeIpXf_00228] [SWS_SomeIpXf_00229] [SWS_SomeIpXf_00232] [SWS_SomeIpXf_00239] [SWS_SomeIpXf_00244] [SWS_SomeIpXf_00300] [SWS_SomeIpXf_00303] [SWS_SomeIpXf_91001]

### B.5.3 Deleted Specification Items in R22-11

[SWS_SomeIpXf_00001] [SWS_SomeIpXf_00002] [SWS_SomeIpXf_00005] [SWS_SomeIpXf_00006] [SWS_SomeIpXf_00007] [SWS_SomeIpXf_00009] [SWS_SomeIpXf_00010] [SWS_SomeIpXf_00011] [SWS_SomeIpXf_00130] [SWS_SomeIpXf_00131] [SWS_SomeIpXf_00132] [SWS_SomeIpXf_00133] [SWS_SomeIpXf_00134]

# C Referenced Meta Classes

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*** | **ApplicationArrayDataType** | | | |
|---|---|---|---|---|
| ***Note*** | An application data type which is an array, each element is of the same application data type.<br>**Tags:** atp.recommendedPackage=ApplicationDataTypes | | | |
| ***Base*** | *ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| dynamicArraySizeProfile | String | 0..1 | attr | Specifies the profile which the array will follow if it is a variable size array. |
| element | ApplicationArrayElement | 0..1 | aggr | This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine.<br>**Tags:** atp.Splitkey=element, element.shortName |

**Table C.1: ApplicationArrayDataType**

| ***Class*** | **ApplicationError** | | | |
|---|---|---|---|---|
| ***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* | | | |
| ***Aggregated by*** | ClientServerInterface.possibleError | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| errorCode | Integer | 0..1 | attr | The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification). |

**Table C.2: ApplicationError**

| ***Class*** | **ApplicationPrimitiveDataType** | | | |
|---|---|---|---|---|
| ***Note*** | A primitive data type defines a set of allowed values.<br>**Tags:** atp.recommendedPackage=ApplicationDataTypes | | | |
| ***Base*** | *ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| – | – | – | – | – |

**Table C.3: ApplicationPrimitiveDataType**

| *Class* | **ArgumentDataPrototype** | | | |
|---|---|---|---|---|
| *Note* | An argument of an operation, carries direction and implementation information. | | | |
| *Base* | *ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable* | | | |
| *Aggregated by* | *AtpClassifier*.atpFeature, ClientServerOperation.argument | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| direction | ArgumentDirection Enum | 0..1 | attr | This attribute specifies the direction of the argument. |
| serverArgument ImplPolicy | ServerArgumentImpl PolicyEnum | 0..1 | attr | This defines how the argument type of the servers `RunnableEntity` is implemented.<br>If the attribute is not defined this has the same semantics as if the attribute is set to the value `useArgumentType` for primitive arguments and structures. |

**Table C.4: ArgumentDataPrototype**

| *Enumeration* | **ArraySizeSemanticsEnum** |
|---|---|
| *Note* | This enumeration provides options for how the number of elements in an array object shall be interpreted. |
| *Aggregated by* | ApplicationArrayElement.arraySizeSemantics, DiagnosticDataElement.arraySizeSemantics, ImplementationDataTypeElement.arraySizeSemantics, SwTextProps.arraySizeSemantics |
| *Literal* | *Description* |
| fixedSize | This means that the array object will always have a fixed number of elements.<br>**Tags:** atp.EnumerationLiteralIndex=0 |
| variableSize | This implies that the actual number of elements in the array object might vary at run-time. The configured size represents the maximum number of elements in the array object.<br>**Tags:** atp.EnumerationLiteralIndex=1 |

**Table C.5: ArraySizeSemanticsEnum**

| *Class* | ***AutosarDataType*** (abstract) | | | |
|---|---|---|---|---|
| *Note* | Abstract base class for user defined AUTOSAR data types for software. | | | |
| *Base* | *ARElement*, *ARObject*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable* | | | |
| *Subclasses* | *AbstractImplementationDataType*, *ApplicationDataType* | | | |
| *Aggregated by* | ARPackage.element | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| swDataDef Props | SwDataDefProps | 0..1 | aggr | The properties of this AutosarDataType.<br>**Tags:** atp.Splitkey=swDataDefProps |

**Table C.6: AutosarDataType**

| *Class* | ***BaseType*** (abstract) | | | |
|---|---|---|---|---|
| *Note* | This abstract meta-class represents the ability to specify a platform dependent base type. | | | |
| *Base* | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable* | | | |
| *Subclasses* | SwBaseType | | | |
| *Aggregated by* | ARPackage.element | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |

▽

△

| ***Class*** | ***BaseType*** (abstract) | | | |
|---|---|---|---|---|
| baseType Definition | BaseTypeDefinition | 1 | aggr | This is the actual definition of the base type.<br>**Tags:**<br>xml.roleElement=false<br>xml.roleWrapperElement=false<br>xml.sequenceOffset=20<br>xml.typeElement=false<br>xml.typeWrapperElement=false |

**Table C.7: BaseType**

| ***Class*** | **BaseTypeDirectDefinition** | | | |
|---|---|---|---|---|
| ***Note*** | This BaseType is defined directly (as opposite to a derived BaseType) | | | |
| ***Base*** | *ARObject*, *BaseTypeDefinition* | | | |
| ***Aggregated by*** | *BaseType*.baseTypeDefinition | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| baseType Encoding | BaseTypeEncoding String | 0..1 | attr | This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence.<br>**Tags:** xml.sequenceOffset=90 |
| baseTypeSize | PositiveInteger | 0..1 | attr | Describes the length of the data type specified in the container in bits.<br>**Tags:** xml.sequenceOffset=70 |
| byteOrder | ByteOrderEnum | 0..1 | attr | This attribute specifies the byte order of the base type.<br>**Tags:** xml.sequenceOffset=110 |
| memAlignment | PositiveInteger | 0..1 | attr | This attribute describes the alignment of the memory object in bits. E.g. "8" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. If the value is set to "0" the meaning shall be interpreted as "unspecified".<br>**Tags:** xml.sequenceOffset=100 |
| native Declaration | NativeDeclarationString | 0..1 | attr | This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example<br>BaseType with shortName: "MyUnsignedInt" native Declaration: "unsigned short"<br>Results in<br>typedef unsigned short MyUnsignedInt;<br>If the attribute is not defined the referring Implementation DataTypes will not be generated as a typedef by RTE.<br>If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseType Size.<br>This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems.<br>**Tags:** xml.sequenceOffset=120 |

**Table C.8: BaseTypeDirectDefinition**

| *Class* | **ClientServerInterface** | | | |
|---|---|---|---|---|
| ***Note*** | A client/server interface declares a number of operations that can be invoked on a server by a client.<br>**Tags:** atp.recommendedPackage=PortInterfaces | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| operation | ClientServerOperation | * | aggr | `ClientServerOperation`(s) of this `ClientServerInterface`.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=operation, operation.shortName, operation.variationPoint.shortLabel<br>vh.latestBindingTime=blueprintDerivationTime<br>This Attribute is only used by the AUTOSAR Classic Platform. |
| possibleError | ApplicationError | * | aggr | Application errors that are defined as part of this interface.<br>**Tags:** atp.Splitkey=possibleError, possibleError.shortName |

**Table C.9: ClientServerInterface**

| *Class* | **ClientServerOperation** | | | |
|---|---|---|---|---|
| ***Note*** | An operation declared within the scope of a client/server interface. | | | |
| ***Base*** | *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Aggregated by*** | ApplicationInterface.command, *AtpClassifier*.atpFeature, ClientServerInterface.operation, DiagnosticDataElementInterface.read, DiagnosticDataIdentifierInterface.read, DiagnosticDataIdentifierInterface.write, DiagnosticExtendedDataRecordInterface.provide, DiagnosticRoutineInterface.requestResult, DiagnosticRoutineInterface.start, DiagnosticRoutineInterface.stop, ServiceInterface.method | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| argument (ordered) | ArgumentDataPrototype | * | aggr | An argument of this `ClientServerOperation`.<br>**Stereotypes:** atpVariation<br>**Tags:** vh.latestBindingTime=blueprintDerivationTime |
| diagArgIntegrity | Boolean | 0..1 | attr | This attribute shall only be used in the implementation of diagnostic routines to support the case where input and output arguments are allocated in a shared buffer and might unintentionally overwrite input arguments by tentative write operations to output arguments.<br>This situation can happen during sliced execution or while output parameters are arrays (call by reference). The value true means that the `ClientServerOperation` is aware of the usage of a shared buffer and takes precautions to avoid unintentional overwrite of input arguments.<br>If the attribute does not exist or is set to false the `ClientServerOperation` does not have to consider the usage of a shared buffer.<br>This Attribute is only used by the AUTOSAR Classic Platform. |
| possibleError | ApplicationError | * | ref | Possible errors that may by raised by the referring operation.<br>**Tags:** atp.Splitkey=possibleError<br>This Attribute is only used by the AUTOSAR Classic Platform. |

**Table C.10: ClientServerOperation**

| *Class* | **ClientServerToSignalMapping** | | | |
|---|---|---|---|---|
| ***Note*** | This element maps the ClientServerOperation to call- and return-SystemSignals. | | | |
| ***Base*** | *ARObject*, *DataMapping* | | | |
| ***Aggregated by*** | SystemMapping.dataMapping | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| callSignal | SystemSignal | 0..1 | ref | Reference to the callSignal to which the IN and INOUT ArgumentDataPrototypes are mapped.<br>**Tags:** atp.Splitkey=callSignal |
| clientServer Operation | ClientServerOperation | 0..1 | iref | Reference to a ClientServerOperation, which is mapped to a call SystemSignal and a return SystemSignal.<br>**Tags:** atp.Splitkey=clientServerOperation, clientServer Operation.contextComposition, clientServer Operation.contextComponent, clientServer Operation.contextPort, clientServerOperation.target Operation<br>**InstanceRef implemented by:** OperationInSystem InstanceRef |
| returnSignal | SystemSignal | 0..1 | ref | Reference to the returnSignal to which the OUT and INOUT ArgumentDataPrototypes are mapped.<br>**Tags:** atp.Splitkey=returnSignal |

**Table C.11: ClientServerToSignalMapping**

| *Class* | ***DataPrototype*** (abstract) | | | |
|---|---|---|---|---|
| ***Note*** | Base class for prototypical roles of any data type. | | | |
| ***Base*** | *ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Subclasses*** | *ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype* | | | |
| ***Aggregated by*** | *AtpClassifier*.atpFeature | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| swDataDef Props | SwDataDefProps | 0..1 | aggr | This property allows to specify data definition properties which apply on data prototype level.<br>**Tags:** atp.Splitkey=swDataDefProps |

**Table C.12: DataPrototype**

| *Class* | **DataTransformation** | | | |
|---|---|---|---|---|
| ***Note*** | A DataTransformation represents a transformer chain. It is an ordered list of transformers. | | | |
| ***Base*** | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Aggregated by*** | DataTransformationSet.dataTransformation | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| data Transformation Kind | DataTransformationKind Enum | 0..1 | attr | This attribute controls the kind of DataTransformation to be applied. |
| executeDespite Data Unavailability | Boolean | 0..1 | attr | Specifies whether the transformer chain is executed even if no input data are available. |
| transformer Chain (ordered) | Transformation Technology | * | ref | This attribute represents the definition of a chain of transformers that are supposed to be executed according to the order of being referenced from DataTransformation. |

**Table C.13: DataTransformation**

| *Enumeration* | **DataTransformationErrorHandlingEnum** |
|---|---|
| ***Note*** | This enumeration defines different ways how a RunnableEntity shall handle transformer errors. |
| ***Aggregated by*** | PortAPIOption.errorHandling |
| ***Literal*** | ***Description*** |
| noTransformerError Handling | A runnable does not handle transformer errors.<br>**Tags:** atp.EnumerationLiteralIndex=0 |
| transformerError Handling | The runnable implements the handling of transformer errors.<br>**Tags:** atp.EnumerationLiteralIndex=1 |

**Table C.14: DataTransformationErrorHandlingEnum**

| *Class* | **EcucModuleDef** | | | |
|---|---|---|---|---|
| ***Note*** | Used as the top-level element for configuration definition for Software Modules, including BSW and RTE as well as ECU Infrastructure.<br>**Tags:** atp.recommendedPackage=EcucDefs<br>This Class is only used by the AUTOSAR Classic Platform. | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpDefinition*, *CollectableElement*, *EcucDefinitionElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| apiServicePrefix | CIdentifier | 0..1 | attr | For modules where several instances of the VSMD can be defined the apiServicePrefix defines the API namespace of the derived instances, e.g. Cdd, Xfrm (ComXf, SomeIpXf, E2EXf). |
| container | EcucContainerDef | * | aggr | Aggregates the top-level container definitions of this specific module definition.<br>**Tags:**<br>atp.Splitkey=container, container.shortName<br>xml.sequenceOffset=11 |
| postBuild Loadable Support | Boolean | 0..1 | attr | Indicates if the module supports changing the configuration by loading a new configuration binary at postBuild time. TRUE means yes, FALSE means no. |
| postBuildVariant Support | Boolean | 0..1 | attr | Indicates if a module supports different post-build variants (previously known as post-build selectable configuration sets). TRUE means yes, FALSE means no. |
| refinedModule Def | EcucModuleDef | 0..1 | ref | Optional reference from the Vendor Specific Module Definition to the Standardized Module Definition it refines. In case this EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION this reference shall not be provided. In case this EcucModuleDef has the category VENDOR_SPECIFIC_MODULE_ DEFINITION this reference is mandatory.<br>**Stereotypes:** atpUriDef<br>**Tags:** atp.Splitkey=refinedModuleDef |
| supported ConfigVariant | EcucConfiguration VariantEnum | * | attr | Specifies which ConfigurationVariants are supported by this software module. This attribute is optional if the Ecuc ModuleDef has the category STANDARDIZED_ MODULE_DEFINITION. If the category attribute of the EcucModuleDef is set to VENDOR_SPECIFIC_ MODULE_DEFINITION then this attribute is mandatory.<br>**Tags:** atp.Status=obsolete |

**Table C.15: EcucModuleDef**

| ***Class*** | **ISignal** | | | |
|---|---|---|---|---|
| ***Note*** | Signal of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal is sent in different SignalIPdus to multiple receivers.<br>To support the RTE "signal fan-out" each SignalIPdu contains ISignals. If the same System Signal is to be mapped into several SignalIPdus there is one ISignal needed for each ISignalToIPduMapping.<br>ISignals describe the Interface between the Precompile configured RTE and the potentially Postbuild configured Com Stack (see ECUC Parameter Mapping).<br>In case of the SystemSignalGroup an ISignal shall be created for each SystemSignal contained in the SystemSignalGroup.<br>**Tags:** atp.recommendedPackage=ISignals | | | |
| ***Base*** | *ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| data Transformation | DataTransformation | 0..1 | ref | Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignal.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=dataTransformation, data Transformation.dataTransformation, data Transformation.variationPoint.shortLabel<br>vh.latestBindingTime=codeGenerationTime |
| dataTypePolicy | DataTypePolicyEnum | 0..1 | attr | With the aggregation of SwDataDefProps an ISignal specifies how it is represented on the network. This representation follows a particular policy. Note that this causes some redundancy which is intended and can be used to support flexible development methodology as well as subsequent integrity checks.<br>If the policy "networkRepresentationFromComSpec" is chosen the network representation from the ComSpec that is aggregated by the PortPrototype shall be used. If the "override" policy is chosen the requirements specified in the PortInterface and in the ComSpec are not fulfilled by the networkRepresentationProps. In case the System Description doesn't use a complete Software Component Description (VFB View) the "legacy" policy can be chosen. |
| initValue | ValueSpecification | 0..1 | aggr | Optional definition of a ISignal's initValue in case the System Description doesn't use a complete Software Component Description (VFB View). This supports the inclusion of legacy system signals.<br>This value can be used to configure the Signal's "Init Value".<br>If a full DataMapping exist for the SystemSignal this information may be available from a configured Sender ComSpec and ReceiverComSpec. In this case the initvalues in SenderComSpec and/or ReceiverComSpec override this optional value specification. Further restrictions apply from the RTE specification.<br>**Tags:** atp.Splitkey=initValue |
| iSignalProps | ISignalProps | 0..1 | aggr | Additional optional ISignal properties that may be stored in different files.<br>**Tags:** atp.Splitkey=iSignalProps |
| iSignalType | ISignalTypeEnum | 0..1 | attr | This attribute defines whether this iSignal is an array that results in a UINT8_N / UINT8_DYN ComSignalType in the COM configuration or a primitive type. |

△

| ***Class*** | **ISignal** | | | |
|---|---|---|---|---|
| length | UnlimitedInteger | 0..1 | attr | Size of the signal in bits. The size needs to be derived from the mapped VariableDataPrototype according to the mapping of primitive DataTypes to BaseTypes as used in the RTE. Indicates maximum size for dynamic length signals.<br>The ISignal length of zero bits is allowed. |
| network Representation Props | SwDataDefProps | 0..1 | aggr | Specification of the actual network representation. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType. The optional baseType attributes "memAllignment" and "byteOrder" shall not be used.<br>The attribute "dataTypePolicy" in the SystemTemplate element defines whether this network representation shall be ignored and the information shall be taken over from the network representation of the ComSpec.<br>If "override" is chosen by the system integrator the network representation can violate against the requirements defined in the PortInterface and in the network representation of the ComSpec.<br>In case that the System Description doesn't use a complete Software Component Description (VFB View) this element is used to configure "ComSignalDataInvalid Value" and the Data Semantics.<br>**Tags:** atp.Splitkey=networkRepresentationProps |
| systemSignal | SystemSignal | 0..1 | ref | Reference to the System Signal that is supposed to be transmitted in the ISignal.<br>**Tags:** atp.Splitkey=systemSignal |
| timeout Substitution Value | ValueSpecification | 0..1 | aggr | Defines and enables the ComTimeoutSubstituition for this ISignal.<br>**Tags:** atp.Splitkey=timeoutSubstitutionValue |
| transformation ISignalProps | TransformationISignal Props | * | aggr | A transformer chain consists of an ordered list of transformers. The ISignal specific configuration properties for each transformer are defined in the TransformationISignalProps class. The transformer configuration properties that are common for all ISignals are described in the TransformationTechnology class.<br>**Tags:** atp.Splitkey=transformationISignalProps |

**Table C.16: ISignal**

| ***Class*** | ***Identifiable*** (abstract) |
|---|---|
| ***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* |

▽

△

| **Class** | **Identifiable** (abstract) | | | |
|---|---|---|---|---|
| **Subclasses** | ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, AppOsTask ProxyToEcuTaskProxyMapping, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartition Mapping, AppliedStandard, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpClassifier*, *Atp Feature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BinaryManifestAddressable Object*, BinaryManifestItemDefinition, *BinaryManifestResource*, BinaryManifestResourceDefinition, Block State, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanSecSecureChannel, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClientIdDefinition, Client ServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, *CouplingElementAbstractDetails*, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPort StructuralElement*, *CpSoftwareClusterResource*, CpSoftwareClusterResourceToApplicationPartition Mapping, CpSoftwareClusterToApplicationPartitionMapping, CpSoftwareClusterToEcuInstanceMapping, CpSoftwareClusterToResourceMapping, *CryptoServiceMapping*, CyclicHandlingComDataToOsTask ProxyMapping, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, DataTransformation, *Dds AbstractServiceInstanceElementCp*, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, Diagnostic ExtendedDataRecordElement, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltArgumentProps, DltLogChannel, Dlt Message, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, ECUMapping, *EOCExecutableEntity RefAbstract*, EcuPartition, EcuPartitionToCoreMapping, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, Ethernet WakeupSleepOnDatalineConfig, EventHandler, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeature Relation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayTpConnection Control, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722 TpAcfBus*, *IEEE1722TpAcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignal Triggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, J1939Node, J1939SharedAddress Cluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacAddress VlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, Mode Declaration, ModeDeclarationMapping, ModeSwitchPoint, ModeSwitchSenderComSpecProps, Network Endpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *Physical Channel*, PortElementToCommunicationResourceMapping, PortGroup, *PortInterfaceMapping*, Queued ReceiverComSpecProps, ResourceConsumption, RootSwCompositionPrototype, RptComponent, Rpt Container, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptService Point, RteEventInCompositionSeparation, RteEventInCompositionToOsTaskProxyMapping, RteEventIn SystemSeparation, RteEventInSystemToOsTaskProxyMapping, RunnableEntityGroup, *SdgAttribute*, Sdg Class, SecOcJobRequirement, SecureCommunicationAuthenticationProps, SecureCommunication FreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServerCallPoint*, ServerComSpecProps, *ServiceNeeds*, SignalServiceTranslationElementProps, SignalServiceTranslation EventProps, SignalServiceTranslationProps, SocketAddress, SomeipTpChannel, *StackUsage*, Static SocketConnection, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SwitchAsynchronousTraffic ShaperGroupEntry, SwitchAtsInstanceEntry, SwitchFlowMeteringEntry, SwitchStreamFilterActionDest PortModification, SwitchStreamFilterEntry, SwitchStreamFilterRule, SwitchStreamGateEntry, Switch StreamIdentification, SystemMapping, SystemSignalGroupToCommunicationResourceMapping, System SignalToCommunicationResourceMapping, TDCpSoftwareClusterMapping, TDCpSoftwareCluster ResourceMapping, TcpOptionFilterList, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipher Suite, TlsCryptoCipherSuiteProps, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, Variable Access, VariationPointProxy, ViewMap, VlanConfig, WaitPoint | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| adminData | AdminData | 0..1 | aggr | This represents the administrative data for the identifiable object.<br>**Tags:**<br>atp.Splitkey=adminData<br>xml.sequenceOffset=-40 |

▽

△

| ***Class*** | ***Identifiable*** (abstract) | | | |
|---|---|---|---|---|
| annotation | Annotation | * | aggr | Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes.<br>**Tags:**<br>atp.Splitkey=annotation<br>xml.sequenceOffset=-25 |
| category | CategoryString | 0..1 | attr | The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints.<br>**Tags:** xml.sequenceOffset=-50 |
| desc | MultiLanguageOverview Paragraph | 0..1 | aggr | This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question.<br>More elaborate documentation, (in particular how the object is built or used) should go to "introduction".<br>**Tags:**<br>atp.Splitkey=desc<br>xml.sequenceOffset=-60 |
| introduction | DocumentationBlock | 0..1 | aggr | This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock.<br>**Tags:**<br>atp.Splitkey=introduction<br>xml.sequenceOffset=-30 |
| uuid | String | 0..1 | attr | The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp.<br>**Tags:** xml.attribute=true |

**Table C.17: Identifiable**

| ***Class*** | **ImplementationDataType** |
|---|---|
| ***Note*** | Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code.<br>**Tags:** atp.recommendedPackage=ImplementationDataTypes |

▽

△

| *Class* | **ImplementationDataType** | | | |
|---|---|---|---|---|
| ***Base*** | *ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Autosar DataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| dynamicArray SizeProfile | String | 0..1 | attr | Specifies the profile which the array will follow in case this data type is a variable size array. |
| isStructWith Optional Element | Boolean | 0..1 | attr | This attribute is only valid if the attribute category is set to STRUCTURE.<br>If set to true, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional. |
| subElement (ordered) | ImplementationData TypeElement | * | aggr | Specifies an element of an array, struct, or union data type.<br>The aggregation of `ImplementationDataTypeElement` is subject to variability with the purpose to support the conditional existence of elements inside a `ImplementationDataType` representing a structure.<br>**Stereotypes:** atpVariation<br>**Tags:** vh.latestBindingTime=preCompileTime |
| symbolProps | SymbolProps | 0..1 | aggr | This represents the SymbolProps for the Implementation DataType.<br>**Tags:** atp.Splitkey=symbolProps, symbolProps.short Name |
| 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 C.18: ImplementationDataType**

| *Class* | **ImplementationDataTypeElement** | | | |
|---|---|---|---|---|
| ***Note*** | Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated.<br>This element either consists of further `subElement`s or it is further defined via its `swDataDefProps`.<br>There are several use cases within the system of `ImplementationDataType`s for such a local declaration:<br>• It can represent the elements of an array, defining the element type and array size<br>• It can represent an element of a struct, defining its type<br>• It can be the local declaration of a debug element. | | | |
| ***Base*** | *ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Aggregated by*** | *AtpClassifier*.atpFeature, ImplementationDataType.subElement, ImplementationDataTypeElement.sub Element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| arrayImplPolicy | ArrayImplPolicyEnum | 0..1 | attr | This attribute controls the implementation of the payload of an array. It shall only be used if the enclosing `ImplementationDataType` constitutes an array. |

▽

△

| *Class* | **ImplementationDataTypeElement** | | | |
|---|---|---|---|---|
| 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.<br>**Stereotypes:** atpVariation<br>**Tags:** vh.latestBindingTime=preCompileTime |
| arraySize Handling | ArraySizeHandling Enum | 0..1 | attr | The way how the size of the array is handled in case of a variable size array. |
| arraySize Semantics | ArraySizeSemantics Enum | 0..1 | attr | This attribute controls the meaning of the value of the array size. |
| isOptional | Boolean | 0..1 | attr | This attribute represents the ability to declare the enclosing `ImplementationDataTypeElement` as optional. This means that, at runtime, the `ImplementationDataTypeElement` may or may not have a valid value and shall therefore be ignored.<br>The underlying runtime software provides means to set the `CppImplementationDataTypeElement` as not valid at the sending end of a communication and determine its validity at the receiving end. |
| subElement (ordered) | ImplementationData TypeElement | * | aggr | Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs").<br>The aggregation of `ImplementationDataTypeElement` is subject to variability with the purpose to support the conditional existence of elements inside a `ImplementationDataType` representing a structure.<br>**Stereotypes:** atpVariation<br>**Tags:** vh.latestBindingTime=preCompileTime |
| swDataDef Props | SwDataDefProps | 0..1 | aggr | The properties of this ImplementationDataTypeElement.<br>**Tags:** atp.Splitkey=swDataDefProps |

**Table C.19: ImplementationDataTypeElement**

| *Class* | ***InternalBehavior*** (abstract) | | | |
|---|---|---|---|---|
| ***Note*** | Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters. | | | |
| ***Base*** | *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Subclasses*** | BswInternalBehavior, SwcInternalBehavior | | | |
| ***Aggregated by*** | *AtpClassifier*.atpFeature | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |

▽

△

| ***Class*** | ***InternalBehavior*** (abstract) | | | |
|---|---|---|---|---|
| constant Memory | ParameterData Prototype | * | aggr | Describes a read only memory object containing characteristic value(s) implemented by this Internal Behavior.<br>The shortName of ParameterDataPrototype has to be equal to the ''C' identifier of the described constant.<br>The characteristic value(s) might be shared between Sw ComponentPrototypes of the same SwComponentType.<br>The aggregation of constantMemory is subject to variability with the purpose to support variability in the software component or module implementations. Typically different algorithms in the implementation are requiring different number of memory objects.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=constantMemory, constantMemory.short Name, constantMemory.variationPoint.shortLabel vh.latestBindingTime=preCompileTime |
| constantValue Mapping | ConstantSpecification MappingSet | * | ref | Reference to the ConstantSpecificationMapping to be applied for the particular InternalBehavior<br>**Tags:** atp.Splitkey=constantValueMapping |
| dataType Mapping | DataTypeMappingSet | * | ref | Reference to the DataTypeMapping to be applied for the particular InternalBehavior<br>**Tags:** atp.Splitkey=dataTypeMapping |
| exclusiveArea | ExclusiveArea | * | aggr | This specifies an ExclusiveArea for this InternalBehavior. The exclusiveArea is local to the component resp. module. The aggregation of ExclusiveAreas is subject to variability. Note: the number of ExclusiveAreas might vary due to the conditional existence of RunnableEntities or BswModuleEntities.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=exclusiveArea, exclusiveArea.shortName, exclusiveArea.variationPoint.shortLabel vh.latestBindingTime=preCompileTime |
| exclusiveArea NestingOrder | ExclusiveAreaNesting Order | * | aggr | This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=exclusiveAreaNestingOrder, exclusiveArea NestingOrder.shortName, exclusiveAreaNesting Order.variationPoint.shortLabel vh.latestBindingTime=preCompileTime |

▽

△

| *Class* | *InternalBehavior* (abstract) | | | |
|---|---|---|---|---|
| staticMemory | VariableDataPrototype | * | aggr | Describes a read and writeable static memory object representing measurerment variables implemented by this software component. The term "static" is used in the meaning of "non-temporary" and does not necessarily specify a linker encapsulation. This kind of memory is only supported if supportsMultipleInstantiation is FALSE.<br>The shortName of the VariableDataPrototype has to be equal with the ''C' identifier of the described variable.<br>The aggregation of staticMemory is subject to variability with the purpose to support variability in the software component's implementations.<br>Typically different algorithms in the implementation are requiring different number of memory objects.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=staticMemory, staticMemory.shortName, staticMemory.variationPoint.shortLabel<br>vh.latestBindingTime=preCompileTime |

**Table C.20: InternalBehavior**

| *Class* | **PortAPIOption** | | | |
|---|---|---|---|---|
| *Note* | Options how to generate the signatures of calls for an AtomicSwComponentType in order to communicate over a PortPrototype (for calls into a RunnableEntity as well as for calls from a Runnable Entity to the PortPrototype). | | | |
| *Base* | *ARObject* | | | |
| *Aggregated by* | SwcInternalBehavior.portAPIOption | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| enableTake Address | Boolean | 0..1 | attr | If set to true, the software-component is able to use the API reference for deriving a pointer to an object. |
| errorHandling | DataTransformation ErrorHandlingEnum | 0..1 | attr | This specifies whether a RunnableEntity accessing a Port Prototype that is referenced by this PortAPIOption shall specifically handle transformer errors or not. |
| indirectAPI | Boolean | 0..1 | attr | If set to true this attribute specifies an "indirect API" to be generated for the associated port which means that the software-component is able to access the actions on a port via a pointer to an object representing a port. This allows e.g. iterating over ports in a loop. This option has no effect for PPortPrototypes of client/server interfaces. |
| port | PortPrototype | 0..1 | ref | The option is valid for generated functions related to communication over this port<br>**Stereotypes:** atpIdentityContributor |
| portArgValue (ordered) | PortDefinedArgument Value | * | aggr | An argument value defined by this port. |
| supported Feature | SwcSupportedFeature | * | aggr | This collection specifies which features are supported by the RunnableEntitys which access a PortPrototype that it referenced by this PortAPIOption.<br>**Tags:** atp.Splitkey=supportedFeature |
| transformer Status Forwarding | DataTransformation StatusForwardingEnum | 0..1 | attr | This attribute specifies whether a RunnableEntity accessing a PortPrototype that is referenced by this Port APIOption shall be able to forward a status code to the transformer chain. |

**Table C.21: PortAPIOption**

| Class | PortDefinedArgumentValue | | | |
|---|---|---|---|---|
| Note | A PortDefinedArgumentValue is passed to a RunnableEntity dealing with the ClientServerOperations provided by a given PortPrototype. Note that this is restricted to PPortPrototypes of a ClientServer Interface. | | | |
| Base | *ARObject* | | | |
| Aggregated by | PortAPIOption.portArgValue | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| value | ValueSpecification | 0..1 | aggr | Specifies the actual value. |
| valueType | ImplementationDataType | 0..1 | tref | The implementation type of this argument value. It should not be composite type or a pointer.<br>**Stereotypes:** isOfType |

**Table C.22: PortDefinedArgumentValue**

| Class | SOMEIPTransformationProps | | | |
|---|---|---|---|---|
| Note | The class SOMEIPTransformationProps specifies SOME/IP specific configuration properties. | | | |
| Base | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TransformationProps* | | | |
| Aggregated by | TransformationPropsSet.transformationProps | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| alignment | PositiveInteger | 0..1 | attr | Defines the padding for alignment purposes that will be added by the SOME/IP transformer after the serialized data of the variable data length data element. The alignment shall be specified in Bits. |
| sizeOfArrayLengthField | PositiveInteger | 0..1 | attr | This attribute describes the size of the length field (in Bytes) that will be put in front of the referenced Array in the SOME/IP message. |
| sizeOfStringLengthField | PositiveInteger | 0..1 | attr | This attribute describes the size of the length field (in Bytes) that will be put in front of the referenced String in the SOME/IP message. |
| sizeOfStructLengthField | PositiveInteger | 0..1 | attr | This attribute describes the size of the length field (in Bytes) that will be put in front of a Structure in the SOME/IP message. |
| sizeOfUnionLengthField | PositiveInteger | 0..1 | attr | This attribute describes the size of the length field (in Bytes) that will be put in front of a Union in the SOME/IP message. |

**Table C.23: SOMEIPTransformationProps**

| Class | SenderReceiverInterface | | | |
|---|---|---|---|---|
| Note | A sender/receiver interface declares a number of data elements to be sent and received.<br>**Tags:** atp.recommendedPackage=PortInterfaces | | | |
| Base | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| Aggregated by | ARPackage.element | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| dataElement | VariableDataPrototype | * | aggr | The data elements of this `SenderReceiverInterface`.<br>**Tags:** atp.Splitkey=dataElement, dataElement.shortName |
| invalidationPolicy | InvalidationPolicy | * | aggr | InvalidationPolicy for a particular dataElement<br>**Tags:** atp.Splitkey=invalidationPolicy |

▽

△

| Class | SenderReceiverInterface | | | |
|---|---|---|---|---|
| metaDataItem Set | MetaDataItemSet | * | aggr | This aggregation defines fixed sets of meta-data items associated with `dataElement`s of the enclosing `SenderReceiverInterface`<br>**Tags:** atp.Splitkey=metaDataItemSet |

**Table C.24: SenderReceiverInterface**

| Class | SenderReceiverToSignalMapping | | | |
|---|---|---|---|---|
| **Note** | Mapping of a sender receiver communication data element to a signal. | | | |
| **Base** | *ARObject*, *DataMapping* | | | |
| **Aggregated by** | SystemMapping.dataMapping | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| dataElement | VariableDataPrototype | 0..1 | iref | Reference to the data element.<br>**Tags:** atp.Splitkey=dataElement, dataElement.context Component, dataElement.contextComposition, data Element.contextPort, dataElement.targetDataPrototype<br>**InstanceRef implemented by:** VariableDataPrototypeIn SystemInstanceRef |
| senderToSignal TextTable Mapping | TextTableMapping | 0..1 | aggr | This mapping allows for the text-table translation between the sending DataPrototype that is defined in the Port Prototype and the physicalProps defined for the System Signal.<br>**Tags:** atp.Splitkey=senderToSignalTextTableMapping |
| signalTo ReceiverText TableMapping | TextTableMapping | 0..1 | aggr | This mapping allows for the text-table translation between the physicalProps defined for the SystemSignal and a receiving DataPrototype that is defined in the Port Prototype.<br>**Tags:** atp.Splitkey=signalToReceiverTextTableMapping |
| systemSignal | SystemSignal | 0..1 | ref | Reference to the system signal used to carry the data element.<br>**Tags:** atp.Splitkey=systemSignal |

**Table C.25: SenderReceiverToSignalMapping**

| Class | SwBaseType | | | |
|---|---|---|---|---|
| **Note** | This meta-class represents a base type used within ECU software.<br>**Tags:** atp.recommendedPackage=BaseTypes | | | |
| **Base** | *ARElement*, *ARObject*, *AtpBlueprint*, *BaseType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable* | | | |
| **Aggregated by** | ARPackage.element | | | |
| **Attribute** | **Type** | **Mult.** | **Kind** | **Note** |
| – | – | – | – | – |

**Table C.26: SwBaseType**

| ***Class*** | «atpVariation» **SwDataDefProps** | | | |
|---|---|---|---|---|
| ***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.<br><br>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.<br><br>SwDataDefProps covers various aspects:<br>• 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<br>• Implementation aspects, mainly expressed by swImplPolicy, swAddrMethod, baseType, implementationDataType and additionalNativeTypeQualifier<br>• Access policy for the MCD system, mainly expressed by swCalibrationAccess<br>• Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalid Value<br>• Code generation policy provided by swRecordLayout<br>**Tags:** vh.latestBindingTime=codeGenerationTime | | | |
| ***Base*** | *ARObject* | | | |
| ***Aggregated by*** | *AutosarDataType*.swDataDefProps, CompositeNetworkRepresentation.networkRepresentation, CppImplementationDataTypeElement.swDataDefProps, *DataPrototype*.swDataDefProps, DataPrototypeTransformationProps.networkRepresentationProps, DiagnosticDataElement.swDataDefProps, DiagnosticEnvDataElementCondition.swDataDefProps, DiagnosticExtendedDataRecordElement.swDataDefProps, DiagnosticSovdPrimitiveContentElement.swDataDefProps, DltArgumentProps.networkRep, FlatInstanceDescriptor.swDataDefProps, ImplementationDataTypeElement.swDataDefProps, InstantiationDataDefProps.swDataDefProps, ISignal.networkRepresentationProps, McDataInstance.resultingProperties, ParameterAccess.swDataDefProps, PerInstanceMemory.swDataDefProps, *ReceiverComSpec*.networkRepresentation, SecurityEventContextDataElement.networkRepresentation, *SenderComSpec*.networkRepresentation, SomeipDataPrototypeTransformationProps.networkRepresentation, SwPointerTargetProps.swDataDefProps, SwServiceArg.swDataDefProps, SwSystemconst.swDataDefProps, SystemSignal.physicalProps | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| additionalNativeTypeQualifier | NativeDeclarationString | 0..1 | attr | This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string.<br><br>**Tags:** xml.sequenceOffset=235 |
| annotation | Annotation | * | aggr | This aggregation allows to add annotations (yellow pads ...) related to the current data object.<br><br>**Tags:**<br>atp.Splitkey=annotation<br>xml.roleElement=true<br>xml.roleWrapperElement=true<br>xml.sequenceOffset=20<br>xml.typeElement=false<br>xml.typeWrapperElement=false |
| baseType | SwBaseType | 0..1 | ref | Base type associated with the containing data object.<br><br>**Tags:**<br>atp.Splitkey=baseType<br>xml.sequenceOffset=50 |

▽

△

| *Class* | «atpVariation» **SwDataDefProps** | | | |
|---|---|---|---|---|
| compuMethod | CompuMethod | 0..1 | ref | Computation method associated with the semantics of this data object.<br>**Tags:**<br>atp.Splitkey=compuMethod<br>xml.sequenceOffset=180 |
| dataConstr | DataConstr | 0..1 | ref | Data constraint for this data object.<br>**Tags:**<br>atp.Splitkey=dataConstr<br>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.<br>**Tags:** xml.sequenceOffset=210 |
| display Presentation | DisplayPresentation Enum | 0..1 | attr | This attribute controls the presentation of the related data for measurement and calibration tools. |
| implementation DataType | AbstractImplementation DataType | 0..1 | ref | This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially<br>• redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype<br>• the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly<br>• the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly<br>• the data type of an SwServiceArg, if it does not refer to a base type directly<br>**Tags:**<br>atp.Splitkey=implementationDataType<br>xml.sequenceOffset=215 |
| invalidValue | ValueSpecification | 0..1 | aggr | Optional value to express invalidity of the actual data element.<br>**Tags:**<br>atp.Splitkey=invalidValue<br>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.<br>**Tags:**<br>atp.Splitkey=swAddrMethod<br>xml.sequenceOffset=30 |
| swAlignment | AlignmentType | 0..1 | attr | The attribute describes the intended typical alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced Sw AddrMethod.<br>**Tags:** xml.sequenceOffset=33 |

▽

△

| *Class* | «atpVariation» **SwDataDefProps** | | | |
|---|---|---|---|---|
| swBit Representation | SwBitRepresentation | 0..1 | aggr | Description of the binary representation in case of a bit variable.<br>**Tags:**<br>atp.Splitkey=swBitRepresentation<br>xml.sequenceOffset=60 |
| swCalibration Access | SwCalibrationAccess Enum | 0..1 | attr | Specifies the read or write access by MCD tools for this data object.<br>**Tags:** xml.sequenceOffset=70 |
| swCalprmAxis Set | SwCalprmAxisSet | 0..1 | aggr | This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters.<br>**Tags:**<br>atp.Splitkey=swCalprmAxisSet<br>xml.sequenceOffset=90 |
| swComparison Variable | SwVariableRefProxy | * | aggr | Variables used for comparison in an MCD process.<br>**Tags:**<br>atp.Splitkey=swComparisonVariable<br>xml.sequenceOffset=170<br>xml.typeElement=false |
| swData Dependency | SwDataDependency | 0..1 | aggr | Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system).<br>**Tags:**<br>atp.Splitkey=swDataDependency<br>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.<br>**Tags:**<br>atp.Splitkey=swHostVariable<br>xml.sequenceOffset=220<br>xml.typeElement=false |
| swImplPolicy | SwImplPolicyEnum | 0..1 | attr | Implementation policy for this data object.<br>**Tags:** xml.sequenceOffset=230 |
| swIntended Resolution | Numerical | 0..1 | attr | The purpose of this element is to describe the requested quantization of data objects early on in the design process.<br>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).<br>In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution.<br>The resolution is specified in the physical domain according to the property "unit".<br>**Tags:** xml.sequenceOffset=240 |
| swInterpolation Method | Identifier | 0..1 | attr | This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked.<br>**Tags:** xml.sequenceOffset=250 |

▽

△

| *Class* | «atpVariation» **SwDataDefProps** | | | |
|---|---|---|---|---|
| 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 .<br>**Tags:** xml.sequenceOffset=260 |
| swPointerTarget Props | SwPointerTargetProps | 0..1 | aggr | Specifies that the containing data object is a pointer to another data object.<br>**Tags:**<br>atp.Splitkey=swPointerTargetProps<br>xml.sequenceOffset=280 |
| swRecord Layout | SwRecordLayout | 0..1 | ref | Record layout for this data object.<br>**Tags:**<br>atp.Splitkey=swRecordLayout<br>xml.sequenceOffset=290 |
| swRefresh Timing | MultidimensionalTime | 0..1 | aggr | This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system.<br>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.<br>**Tags:**<br>atp.Splitkey=swRefreshTiming<br>xml.sequenceOffset=300 |
| swTextProps | SwTextProps | 0..1 | aggr | the specific properties if the data object is a text object.<br>**Tags:**<br>atp.Splitkey=swTextProps<br>xml.sequenceOffset=120 |
| swValueBlock Size | Numerical | 0..1 | attr | This represents the size of a Value Block<br>**Stereotypes:** atpVariation<br>**Tags:**<br>vh.latestBindingTime=preCompileTime<br>xml.sequenceOffset=80 |
| swValueBlock SizeMult (ordered) | Numerical | * | attr | This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension.<br>The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on.<br>For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist.<br>**Stereotypes:** atpVariation<br>**Tags:** vh.latestBindingTime=preCompileTime |
| unit | Unit | 0..1 | ref | Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible.<br>**Tags:**<br>atp.Splitkey=unit<br>xml.sequenceOffset=350 |

▽

△

| *Class* | «atpVariation» **SwDataDefProps** | | | |
|---|---|---|---|---|
| valueAxisData Type | ApplicationPrimitive DataType | 0..1 | ref | The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType.<br>**Tags:**<br>atp.Splitkey=valueAxisDataType<br>xml.sequenceOffset=355 |

**Table C.27: SwDataDefProps**

| *Class* | **SwTextProps** | | | |
|---|---|---|---|---|
| ***Note*** | This meta-class expresses particular properties applicable to strings in variables or calibration parameters. | | | |
| ***Base*** | *ARObject* | | | |
| ***Aggregated by*** | SwDataDefProps.swTextProps | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| arraySize Semantics | ArraySizeSemantics Enum | 0..1 | attr | This attribute controls the semantics of the arraysize for the array representing the string in an `ImplementationDataType`.<br>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`.<br>**Tags:**<br>atp.Splitkey=baseType<br>xml.sequenceOffset=30 |
| swFillCharacter | Integer | 0..1 | attr | Filler character for text parameter to pad up to the maximum length `swMaxTextSize`.<br>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.<br>The usage of the fill character depends on the `arraySizeSemantics`.<br>**Tags:** xml.sequenceOffset=40 |
| swMaxTextSize | Integer | 0..1 | attr | Specifies the maximum text size in characters. Note the size in bytes depends on the encoding in the corresponding `baseType`.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>vh.latestBindingTime=preCompileTime<br>xml.sequenceOffset=20 |

**Table C.28: SwTextProps**

| *Class* | **SystemSignal** | | | |
|---|---|---|---|---|
| ***Note*** | The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with exactly one system signal defined for each data element prototype sent and received by connected SW component instances.<br>**Tags:** atp.recommendedPackage=SystemSignals | | | |
| ***Base*** | *ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| dynamicLength | Boolean | 0..1 | attr | The length of dynamic length signals is variable in run-time. Only a maximum length of such a signal is specified in the configuration (attribute length in ISignal element). |
| physicalProps | SwDataDefProps | 0..1 | aggr | Specification of the physical representation.<br>**Tags:** atp.Splitkey=physicalProps |

**Table C.29: SystemSignal**

| *Class* | «atpVariation» ***TransformationISignalProps*** (abstract) | | | |
|---|---|---|---|---|
| ***Note*** | TransformationISignalProps holds all the attributes for the different TransformationTechnologies that are ISignal specific.<br>**Tags:** vh.latestBindingTime=postBuild | | | |
| ***Base*** | *ARObject*, *Describable* | | | |
| ***Subclasses*** | DdsTransformationISignalProps, EndToEndTransformationISignalProps, SOMEIPTransformationISignal Props, UserDefinedTransformationISignalProps | | | |
| ***Aggregated by*** | ISignal.transformationISignalProps, ISignalGroup.transformationISignalProps | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| csErrorReaction | CSTransformerError ReactionEnum | 0..1 | attr | Defines whether the transformer chain of client/server communication coordinates an autonomous error reaction together with the RTE or whether any error reaction is the responsibility of the application. |
| dataPrototype Transformation Props | DataPrototype TransformationProps | * | aggr | Fine granular modeling of TransfromationProps on the level of DataPrototypes.<br>**Tags:** atp.Splitkey=dataPrototypeTransformationProps, dataPrototypeTransformationProps.ident.shortName |
| ident | TransformationISignal PropsIdent | 0..1 | aggr | This adds the ability to add a shortName to TransformationISignalProps. Please note that the short-name needs to be provided if the splitable mechanism is used.<br>**Stereotypes:** atpIdentityContributor |
| transformer | Transformation Technology | 0..1 | ref | Reference to the TransformationTechnology description that contains transformer specific and ISignal independent configuration properties.<br>**Tags:** atp.Splitkey=transformer |

**Table C.30: TransformationISignalProps**

| *Class* | **TransformationTechnology** |
|---|---|
| ***Note*** | A TransformationTechnology is a transformer inside a transformer chain.<br>**Tags:** xml.namePlural=TRANSFORMATION-TECHNOLOGIES |
| ***Base*** | *ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable* |

▽

△

| *Class* | **TransformationTechnology** | | | |
|---|---|---|---|---|
| ***Aggregated by*** | DataTransformationSet.transformationTechnology | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| bufferProperties | BufferProperties | 0..1 | aggr | Aggregation of the mandatory BufferProperties.<br>**Tags:** atp.Splitkey=bufferProperties |
| hasInternal State | Boolean | 0..1 | attr | This attribute defines whether the Transformer has an internal state or not. |
| needsOriginal Data | Boolean | 0..1 | attr | Specifies whether this transformer gets access to the SWC's original data. |
| protocol | String | 0..1 | attr | Specifies the protocol that is implemented by this transformer. |
| transformation Description | Transformation Description | 0..1 | aggr | A transformer can be configured with transformer specific parameters which are represented by the Transformer Description.<br>**Stereotypes:** atpVariation<br>**Tags:**<br>atp.Splitkey=transformationDescription, transformation Description.variationPoint.shortLabel<br>vh.latestBindingTime=postBuild |
| transformer Class | TransformerClassEnum | 0..1 | attr | Specifies to which transformer class this transformer belongs. |
| version | String | 0..1 | attr | Version of the implemented protocol. |

**Table C.31: TransformationTechnology**

| *Class* | **TriggerToSignalMapping** | | | |
|---|---|---|---|---|
| ***Note*** | This meta-class represents the ability to map a trigger to a SystemSignal of size 0. The Trigger does not transport any other information than its existence, therefore the limitation in terms of signal length. | | | |
| ***Base*** | *ARObject*, *DataMapping* | | | |
| ***Aggregated by*** | SystemMapping.dataMapping | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| systemSignal | SystemSignal | 0..1 | ref | This is the SystemSignal taken to transport the Trigger over the network.<br>**Tags:**<br>atp.Splitkey=systemSignal<br>xml.sequenceOffset=20 |
| trigger | Trigger | 0..1 | iref | This represents the Trigger that shall be used to trigger RunnableEntities deployed to a remote ECU.<br>**Tags:** atp.Splitkey=trigger, trigger.contextComposition, trigger.contextComponent, trigger.contextPort, trigger.targetTrigger<br>**InstanceRef implemented by:** TriggerInSystemInstance Ref |

**Table C.32: TriggerToSignalMapping**

# D Features of SOME/IP not supported by AUTOSAR SOME/IP transformer

The following features of SOME/IP are currently not supported by the SOME/IP transformer:

- Exceptions and exception-specific error data structures
- Tunneling of SOME/IP messages through CAN and Flexray leads to SOME/IP messages without parts of the header inserted by [4, SWS Socket Adaptor]
- Queued Fire&Forget methods without parameters are not supported by AUTOSAR at all. (Unqueued Fire&Forget methods without parameters and queued Fire&Forget methods with parameters are supported)
- The SOME/IP transformer doesn't check whether variable size arrays contain a minimal number of elements (reason: this is supported by SOME/IP protocol but not by AUTOSAR)
- Optional method arguments: AUTOSAR Classic platform does not support the existence of optional method arguments.

# E Examples

This appendix contains examples which are suitable to help understanding details of the SOME/IP Transformer.

## E.1 Serialization of a Client/Server Operation

As the serialization of inter-ECU Client/Server communication is the most complex scenario, this example will show the resulting APIs which exist in RTE and Transformer both on the Client and the Server as well an overview of the resulting serialized data on the network.

The example deals with two SWCs which are distributed to two ECUs which are connected over some kind of network. The SOME/IP Transformer shall be used to serialize the inter-ECU communication. The client calls a `ClientServerOperation` which is provided by the server. For the server, there are two `PortDefinedArgumentValue`s defined which are applied to the runnable which implements the `ClientServerOperation`. These `PortDefinedArgumentValue`s are only visible within the `InternalBehavior` of the server. They are not visible to the outside world (`ClientServerInterface`) - neither to the client nor in the data on the network.

The following tables define the example `ClientServerInterface` used here.

| | | | |
|---|---|---|---|
| ***Name*** | SomeCSInterface | | |
| ***Comment*** | A ClientServerInterface which contains anything needed to show serialization of ClientServerOperations by SOME/IP Transformer. | | |
| ***IsService*** | false | | |
| ***Variation*** | – | | |
| ***Possible Errors*** | 0 | E_OK | Operation successful |
| | 1 | E_NOT_OK | Operation failed |
| | 2 | E_UNKNOWN_ERROR | An unknown error occurred |

**Table E.1: ClientServerInterface SomeCSInterface**

| | | |
|---|---|---|
| ***Operation*** | SomeCSOperation | |
| ***Comment*** | The ClientServerOperation which is used to demonstrate how the SOME/IP serialization for Client/Sever communication works | |
| ***Mapped to API*** | – | |
| ***Variation*** | – | |
| ***Parameters*** | inputParam1 | |
| | ***Type*** | uint8 |
| | ***Direction*** | IN |
| | ***Comment*** | Refines how source Time Base is cloned to destination |
| | ***Variation*** | – |
| | inputParam2 | |
| | ***Type*** | uint16 |

▽

△

| | | |
|---|---|---|
| | ***Direction*** | IN |
| | ***Comment*** | A parameter which is handed over from the Client to the Server |
| | ***Variation*** | – |
| | biDirectionalParam | |
| | ***Type*** | someStruct |
| | ***Direction*** | INOUT |
| | ***Comment*** | A parameter which is handed over from the Client to the Server, modified by the Server and handed back to the Client |
| | ***Variation*** | – |
| | outputParam1 | |
| | ***Type*** | uint16 |
| | ***Direction*** | OUT |
| | ***Comment*** | A parameter which is handed over from the Server to the Client |
| | ***Variation*** | – |
| | outputParam2 | |
| | ***Type*** | uint32 |
| | ***Direction*** | OUT |
| | ***Comment*** | A parameter which is handed over from the Server to the Client |
| | ***Variation*** | – |
| ***Possible Errors*** | E_OK<br>E_DATA_INCONSISTENT<br>E_UNKNOWN_ERROR | |

**Table E.2: Operation SomeCSOperation**

### E.1.1 Client

On the client side, the following RTE-API is generated according to [SWS_Rte_01102] based on the `ClientServerInterface` which is specified above and the attribute `errorHandling` of `PortAPIOption`:

```
Std_ReturnType Rte_Call_ClientPort_SomeCSOperation
    (uint8 inputParam1,
    uint16 inputParam2,
    someStruct *biDirectionalParam,
    uint16 *outputParam1,
    uint32 *outputParam2,
    Rte_TransformerError *transformerError)
```

For this signature the attribute `errorHandling` of `PortAPIOption` is set to `transformerErrorHandling`. If it would be set to `noTransformerErrorHandling`, the parameter `Rte_TransformerError *transformerError` would not be included in the signature above.

The signature above reflects an synchronous server call. For an asynchronous server call all OUT parameters would be missing for `Rte_Call` but an `Rte_Result` would be necessary instead. The examples for signatures and parameters shown here can be transferred analogously to `Rte_Result`.

This is the API used in the runnable of the client to call the remote server operation.

The RTE executes for the serialization of the request the SOME/IP Transformer with the following API which is specified in [SWS_SomeIpXf_00141]:

```
uint8 SomeIpXf_CSOpSerializer
    (const Rte_Cs_TransactionHandleType *TransactionHandle,
    uint8 *buffer,
    uint16 *bufferLength,
    uint8 inputParam1,
    uint16 inputParam2,
    someStruct biDirectionalParam)
```

This function will serialize the `TransactionHandle` and all IN/INOUT parameters for the request into the following format:

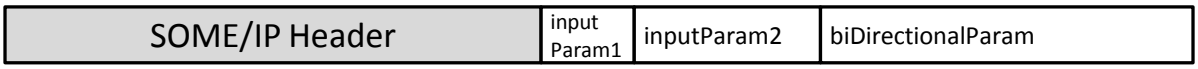


**Figure E.1: Example for serialized data of the Client/Server Request**

The SOME/IP Header contains the TransactionHandle (see [SWS_SomeIpXf_00025] and [SWS_SomeIpXf_00026]).

To deserialize the response that is received by the client after execution of the `ClientServerOperation` on the server the API (according to [SWS_SomeIpXf_00145]) is used:

```
uint8 SomeIpXf_Inv_CSOpSerializer
    (Rte_Cs_TransactionHandleType *TransactionHandle,
    const uint8 *buffer,
    uint16 bufferLength,
    Std_ReturnType *returnValue,
    someStruct *biDirectionalParam,
    uint16 *outputParam1,
    uint32 *outputParam2)
```

### E.1.2 Server

On the server side the `ClientServerOperation` is implemented by a runnable with the following signature which now contains the `PortDefinedArgumentValue`s (see [SWS_Rte_01166]):

```
Std_ReturnType SomeCSOperation
    (uint8 portDefArg1,
    uint8 portDefArg2,
    uint8 inputParam1,
    uint16 inputParam2,
    someStruct *biDirectionalParam,
```

```
    uint16 *outputParam1,
    uint32 *outputParam2)
```

For the deserialization of the received request, the SOME/IP Transformer on the server side, provides according to [SWS_SomeIpXf_00141] this C-API:

```
uint8 SomeIpXf_Inv_CSOpSerializer
    (Rte_Cs_TransactionHandleType *TransactionHandle,
    const uint8 *buffer,
    uint16 bufferLength,
    uint8 *inputParam1,
    uint16 *inputParam2,
    someStruct *biDirectionalParam)
```

The function for serialization of the response is specified by [SWS_SomeIpXf_00145]:

```
uint8 SomeIpXf_CSOpSerializer
    (const Rte_Cs_TransactionHandleType *TransactionHandle,
    uint8 *buffer,
    uint16 *bufferLength,
    Std_ReturnType returnValue,
    someStruct biDirectionalParam,
    uint16 outputParam1,
    uint32 outputParam2)
```

This function will serialize the `TransactionHandle`, the `returnValue` and all IN-OUT/OUT parameters for the response into the following format:

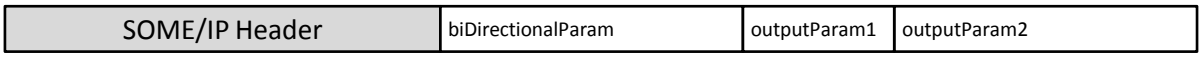


**Figure E.2: Example for serialized data of the Client/Server Response**

The SOME/IP Header contains the `TransactionHandle` and `returnValue` (see [SWS_SomeIpXf_00025], [SWS_SomeIpXf_00026] and [SWS_SomeIpXf_00115]).