Document Title	Specification of SOME/IP Transformer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	660
Document Classification	Standard

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History		
Release	**Changed by**	**Description**
4.2.2	AUTOSAR Release Management	• Size of length fields is configurable • External trigger events are communciated as fire-and-forget methods • Autonomous error reactions of SOME/IP transformer • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
4.2.1	AUTOSAR Release Management	Initial Release

Disclaimer

Table of Contents

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 notfier 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 (empty service is allowed, e.g. for announcing non-SOME/IP services in SOME/IP-SD)
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

3 Related documentation

3.1 Input documents

Bibliography

[1] Glossary
AUTOSAR_TR_Glossary

[2] Specification of Service Discovery
AUTOSAR_SWS_ServiceDiscovery

[3] General Specification on Transformers
AUTOSAR_ASWS_TransformerGeneral

[4] Specification of Socket Adaptor
AUTOSAR_SWS_SocketAdaptor

[5] Specification of RTE Software
AUTOSAR_SWS_RTE

[6] Requirements on AUTOSAR Features
AUTOSAR_RS_Features

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

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

[9] System Template
AUTOSAR_TPS_SystemTemplate

[10] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

3.2 Related standards and norms

Not applicable.

3.3 Related specification

AUTOSAR provides a General Specification on Transformers [3, ASWS 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, ASWS 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 B

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, ASWS Transformer General].

5.1.2 Header file structure

The header file structure of the SOME/IP Transformer is shown in Figure 5.1.

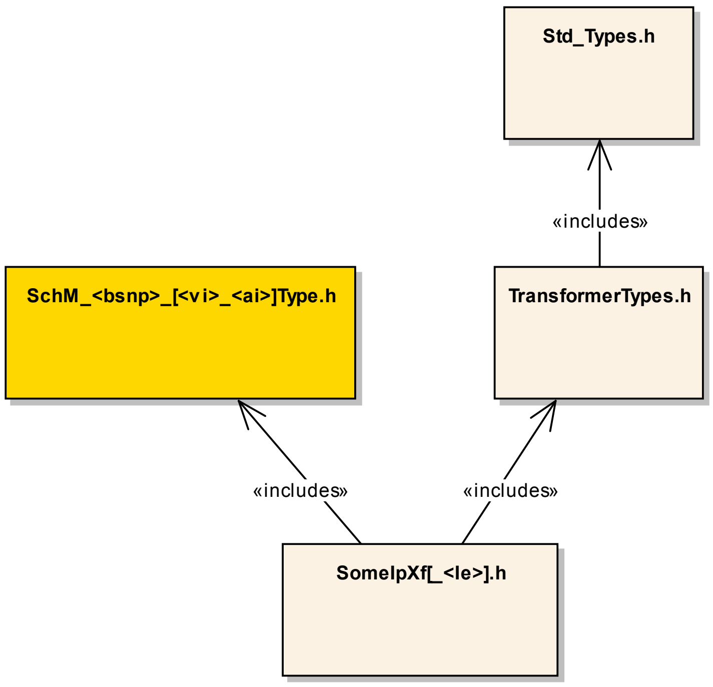

Figure 5.1: Header File Structure of SOME/IP Transformer

[SWS_SomeIpXf_00136] ⌈ The header file `SomeIpXf[_<Ie>].h` shall be the main include file for the SOME/IP transformer and include `TransformerTypes.h` and its Module Interlink Types Header file `SchM_<bsnp>_[<vi>_<ai>]Type.h`.

where
`<Ie>` is the optional implementation specific file name extension according [SWS_BSW_00103],
`<bsnp>` is the BSW Scheduler Name Prefix according [SWS_Rte_07593] and [SWS_Rte_07594],

`<vi>` is the `vendorId` of the BSW module and
`<ai>` is the `vendorApiInfix` of the BSW module. ⌋*(SRS_BSW_00346)*

The file `TransformerTypes.h` contains the general transformer data types.

6 Requirements Tracing

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

Feature	Description	Satisfied by
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_SomeIpXf_00185]
[SRS_BSW_00337]	Classification of development errors	[SWS_SomeIPxf_00184]
[SRS_BSW_00346]	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	[SWS_SomeIpXf_00136]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_SomeIpXf_00183]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_SomeIpXf_00180] [SWS_SomeIpXf_00181] [SWS_SomeIpXf_00182]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_SomeIpXf_00180] [SWS_SomeIpXf_00181] [SWS_SomeIpXf_00182]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_SomeIpXf_00183]
[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]

[SRS_Xfrm_00008]	A transformer shall specify its output format	[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_00013] [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_00105] [SWS_SomeIpXf_00130] [SWS_SomeIpXf_00131] [SWS_SomeIpXf_00132] [SWS_SomeIpXf_00133] [SWS_SomeIpXf_00134] [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]

[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_00035] [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_00151] [SWS_SomeIpXf_00169] [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]
[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_00170] [SWS_SomeIpXf_00176] [SWS_SomeIpXf_00200] [SWS_SomeIpXf_00201] [SWS_SomeIpXf_00202] [SWS_SomeIpXf_00204] [SWS_SomeIpXf_00205]

[SRS_Xfrm_00103]	The SOME/IP Transformer shall support exception notification of applications	[SWS_SomeIpXf_00111]
[SRS_Xfrm_00105]	The SOME/IP Transformer shall support autonomous error reactions on the server side for client/server communication	[SWS_SomeIpXf_00203]

7 Functional specification

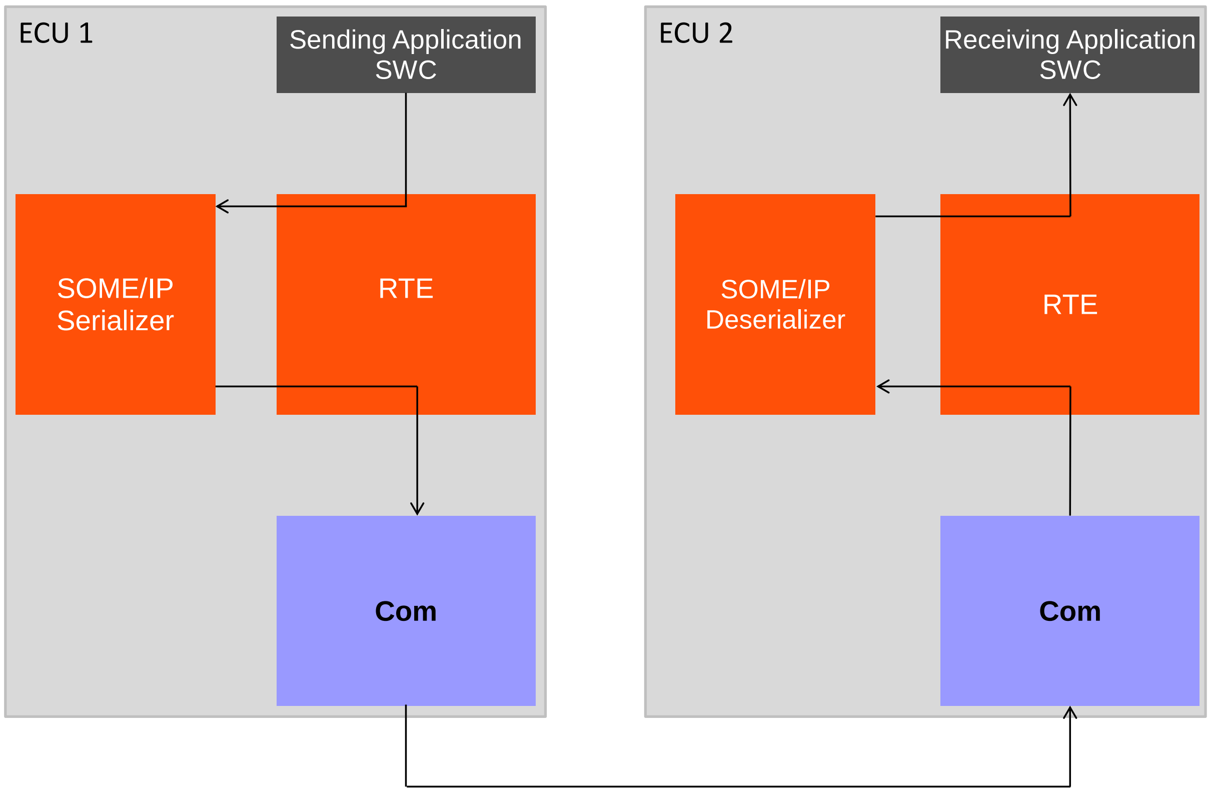

Figure 7.1: Overview of SOME/IP Transformer

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`.

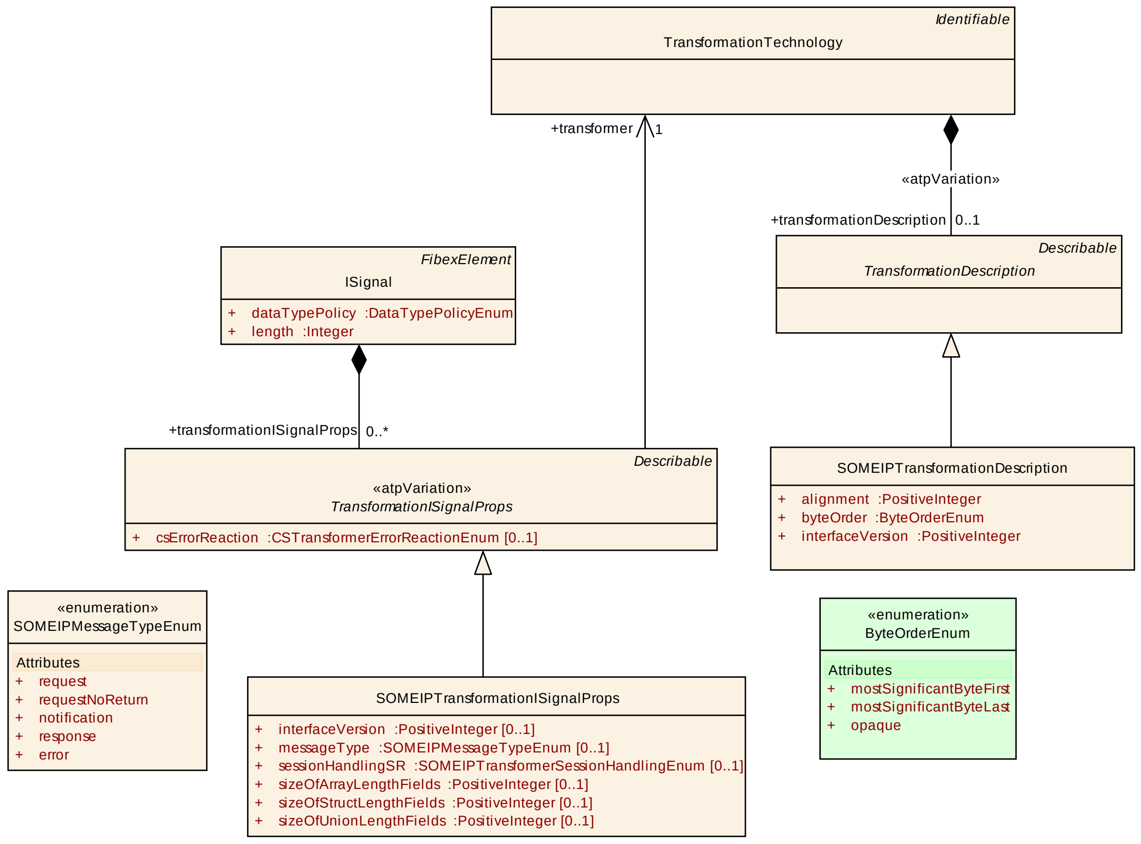
Identifiable
TransformationTechnology
+transformer
1
«atpVariation»
+transformationDescription
0..1
FibexElement
ISignal
+ dataTypePolicy :DataTypePolicyEnum
+ length :Integer
Describable
TransformationDescription
+transformationISignalProps
0..*
Describable
«atpVariation»
TransformationISignalProps
+ csErrorReaction :CSTransformerErrorReactionEnum [0..1]
SOMEIPTransformationDescription
+ alignment :PositiveInteger
+ byteOrder :ByteOrderEnum
+ interfaceVersion :PositiveInteger
«enumeration»
SOMEIPMessageTypeEnum
Attributes
+ request
+ requestNoReturn
+ notification
+ response
+ error
«enumeration»
ByteOrderEnum
Attributes
+ mostSignificantByteFirst
+ mostSignificantByteLast
+ opaque
SOMEIPTransformationISignalProps
+ interfaceVersion :PositiveInteger [0..1]
+ messageType :SOMEIPMessageTypeEnum [0..1]
+ sessionHandlingSR :SOMEIPTransformerSessionHandlingEnum [0..1]
+ sizeOfArrayLengthFields :PositiveInteger [0..1]
+ sizeOfStructLengthFields :PositiveInteger [0..1]
+ sizeOfUnionLengthFields :PositiveInteger [0..1]

Figure 7.2: SOME/IP specific configuration

Class	**SOMEIPTransformationDescription**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	The SOMEIPTransformationDescription is used to specify SOME/IP transformer specific attributes.			
Base	ARObject,Describable,TransformationDescription			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
alignment	PositiveInteger	1	attr	Specifies the alignment of dynamic data in the serialized data stream. The alignment shall be specified in Bits.
byteOrder	ByteOrderEnum	1	attr	Defines which byte order shall be serialized by the SOME/IP transformer
interfaceVersion	PositiveInteger	1	attr	The interface version the SOME/IP transformer shall use.

Table 7.1: SOMEIPTransformationDescription

Class	≪atpVariation≫ **SOMEIPTransformationISignalProps**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	The class SOMEIPTransformationISignalProps specifies ISignal specific configuration properties for SOME/IP transformer attributes.			
Base	ARObject,Describable,TransformationISignalProps			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
interfaceVersion	PositiveInteger	0..1	attr	The interface version the SOME/IP transformer shall use.
messageType	SOMEIPMessageTypeEnum	0..1	attr	The Message Type which shall be placed into the SOME/IP header.
sessionHandlingSR	SOMEIPTransformerSessionHandlingEnum	0..1	attr	Defines whether the SOME/IP transformer shall use session handling for Sender/Receiver communication.
sizeOfArrayLengthFields	PositiveInteger	0..1	attr	The size of all length fields (in Bytes) of fixed-size arrays in the SOME/IP message.
sizeOfStructLengthFields	PositiveInteger	0..1	attr	The size of all length fields (in Bytes) of structs in the SOME/IP message.
sizeOfUnionLengthFields	PositiveInteger	0..1	attr	The size of all length fields (in Bytes) of unions in the SOME/IP message.

Table 7.2: SOMEIPTransformationISignalProps

Enumeration	**ByteOrderEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	When more than one byte is stored in the memory the order of those bytes may differ depending on the architecture of the processing unit. If the least significant byte is stored at the lowest address, this architecture is called little endian and otherwise it is called big endian. ByteOrder is very important in case of communication between different PUs or ECUs.
Literal	***Description***
mostSignificantByteFirst	Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format)
mostSignificantByteLast	Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format)
opaque	For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details.

Table 7.3: ByteOrderEnum

Enumeration	**SOMEIPMessageTypeEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer

Note	Depending on the style of the communication different message types shall be set in the header of a SOME/IP message.
Literal	***Description***
error	The response containing an error.
notification	A request of a notification expecting no response.
request	A request expecting a response.
requestNo Return	A fire&forget request.
response	The response message.

Table 7.4: SOMEIPMessageTypeEnum

[SWS_SomeIpXf_00151] ⌈ 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`

⌋*(SRS_Xfrm_00101)*

7.1 Definition of Identifiers

[SWS_SomeIpXf_00001] ⌈ A service shall be identified using the Service-ID. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00002] ⌈ Service-IDs shall be of type 16 bit length unsigned integer (uint16). ⌋*(SRS_Xfrm_00008)*

The Service-ID of 0xFFFE shall be used to encode non-SOME/IP services. See [SWS_SomeIpXf_00130].

[SWS_SomeIpXf_00005] ⌈ Different services within the same vehicle shall have different Service-IDs. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00006] ⌈ A service instance shall be identified using the Service-Instance-ID. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00007] ⌈ Service-Instance-IDs shall be of type 16 bit length unsigned integer (uint16). ⌋*(SRS_Xfrm_00008)*

The Service-Instance-IDs of 0x0000 and 0xFFFF shall not be used for a service, since 0x0000 is reserved and 0xFFFF is used to describe all service instances. See [SWS_SomeIpXf_00130].

[SWS_SomeIpXf_00009] ⌈ Different service instances within the same vehicle shall have different Service-Instance-IDs.⌋*(SRS_Xfrm_00008)*

Note:
This means that two different camera services shall have two different Service-Instance-IDs SI-ID-1 and SI-ID-2. For all vehicles of a vehicle project SI-ID-1 shall be the same. The same is true for SI-ID-2. If considering another vehicle project, different IDs may be used but it makes sense to use the same IDs among different vehicle projects for ease in testing and integration.

[SWS_SomeIpXf_00010] ⌈ Methods and events shall be identified inside a service using a 16bit Method-ID, which is called Event-ID for events and notifications. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00011] ⌈ Methods shall use Method-IDs with the highest bit set to 0, while the Method-IDs highest bit shall be set to 1 for events and notifications of fields. ⌋*(SRS_Xfrm_00008)*

7.2 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.

7.2.1 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.

7.2.2 Endianess

[SWS_SomeIpXf_00013] ⌈ All headers shall be encoded in network byte order Big Endian (MostSignificantByteFirst) [RFC 791]. ⌋*(SRS_Xfrm_00008)*

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

[SWS_SomeIpXf_00172] ⌈ The byte order of the parameters inside the payload shall be defined by `byteOrder` of `SOMEIPTransformationDescription`. ⌋*(SRS_Xfrm_00008)*

7.2.3 Header

[SWS_SomeIpXf_00152] ⌈ For interoperability reasons the header layout shall be identical for all implementations of SOME/IP and is shown in the Figure 7.3. The fields are presented in transmission order; i.e. the fields on the top left are transmitted first.

In the following sections the different header fields and their usage is being described. ⌋(*SRS_Xfrm_00008*)

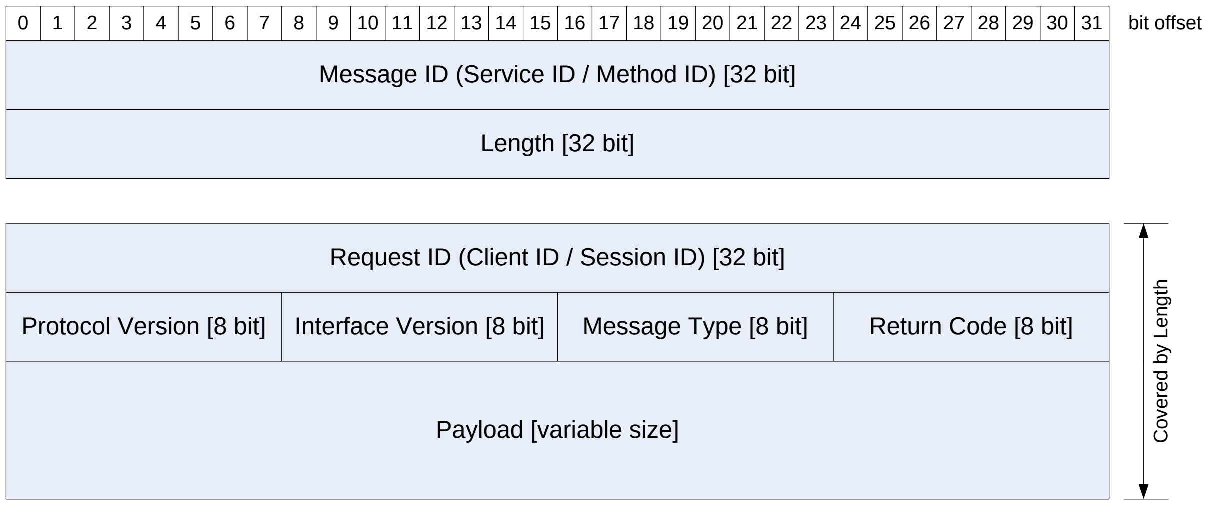

Figure 7.3: SOME/IP Header Format

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

[SWS_SomeIpXf_00015] ⌈ The SOME/IP transformer shall implement all fields of the header except Message ID and Length. ⌋(*SRS_Xfrm_00008*)

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.

7.2.3.1 Message ID [32 bit]

The Message ID is a 32 bit identifier that is used to identify the message. The Message ID has to uniquely identify a method or event of a service.

The assignment of the Message ID is up to the user; however, the Message ID has to be unique for the whole system (i.e. the vehicle). The Message ID can be best compared to a CAN ID and should be handled with a comparable process. The next section 7.2.3.1.1 describes how to structure the Message IDs in order to ease the organization of Message IDs.

7.2.3.1.1 Structure of the Message ID

In order to structure the different methods, events, and fields, they are clustered into services. Services have a set of methods, events, and fields as well as a Service ID, which is only used for this service.

An event shall be part of zero to many eventgroups and an eventgroup shall contain zero to many events. A field shall be part of zero to many eventgroups and an eventgroup can contain zero to many fields.

For inter-ECU Client/Server communication calls we structure the ID in 2^{16} services with 2^{15} methods:

Service ID [16 bit]	0 [1 bit]	Method ID [last 15 bits]

where the 0-Bit is the first bit of the 16 bit Method ID.

With 16 bit Service-ID and a 16 bit Method-ID starting with a 0-Bit (15 bit are still left in the Method-ID for real values), this allows for up to 65536 services with up to 32768 methods each.

Since events and notifications are transported using Client/Server communication, the ID space for the events is further structured:

Service ID [16 bit]	1 [1 bit]	Event ID [last 15 bits]

where the 1-Bit is the first bit of the 16 bit Method ID.

This means that up to 32768 events or notifications per service are possible.

7.2.3.2 Length [32 bit]

The Length field is 32 bit long and contains the length in Byte of the payload beginning with the Request ID/Client ID until the end of the SOME/IP-message.

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

7.2.3.3 Request ID [32 bit]

[SWS_SomeIpXf_00154] ⌈ The Request ID field shall be 32 bit long. ⌋ *(SRS_Xfrm_00008)*

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] ⌈ The Request ID shall be constructed of the Client ID and Session ID:

Client ID [16 bits]	Session ID [16 bits]

⌋ *(SRS_Xfrm_00008)*

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

[SWS_SomeIpXf_00025] ⌈ The `clientId` inside the `Rte_Cs_TransactionHandleType` handed over from RTE shall be used for the value of the Client ID. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00026] ⌈ The `sequenceCounter` inside the `Rte_Cs_TransactionHandleType` handed over from RTE shall be used for the value of the Session ID. ⌋*(SRS_Xfrm_00008)*

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).

7.2.3.4 Protocol Version [8 bit]

[SWS_SomeIpXf_00155] ⌈ The Protocol Version field shall be 8 bit long. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00156] ⌈ The Protocol Version field shall contain the SOME/IP protocol version. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00029] ⌈ The Protocol Version shall be set to 0x01. ⌋*(SRS_Xfrm_00008)*

7.2.3.5 Interface Version [8 bit]

[SWS_SomeIpXf_00030] ⌈ The Interface Version field shall be 8 bit long. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00160] ⌈ The Interface Version field shall contain the Version of the Service Interface. ⌋*(SRS_Xfrm_00008)*

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

7.2.3.6 Message Type [8 bit]

[SWS_SomeIpXf_00161] ⌈ The Message Type field shall be 8 bit long. ⌋(*SRS_Xfrm_00008*)

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

[SWS_SomeIpXf_00031] ⌈ The Message Type field shall be filled with one of the following values:

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

⌋(*SRS_Xfrm_00008*)

A regular client request (message type 0x00) is answered by a server response (message type 0x80), when no error occurred. If errors occur an error message (message type 0x81) will be sent.

For updating values through notification a callback interface exists (message type 0x02).

For Sender/Receiver communication a request is sent that does not have a response message (message type 0x01).

The following values are also valid in SOME/IP in general but are not used by the SOME/IP transformer:

Number	**Value**	**Description**
`0x40`	`REQUEST_ACK`	Acknowledgment for `REQUEST` (optional)
`0x41`	`REQUEST_NO_RETURN_ACK`	Acknowledgment for `REQUEST_NO_RETURN` (informational)
`0x42`	`NOTIFICATION_ACK`	Acknowledgment for `NOTIFICATION` (informational)
`0xC0`	`RESPONSE_ACK`	The Acknowledgment for `RESPONSE` (informational)
`0xC1`	`ERROR_ACK`	Acknowledgment for `ERROR` (informational)

For all messages an optional acknowledgment (ACK) exists for use with transport protocols that do not acknowledge a received message.

7.2.3.7 Return Code [8 bit]

[SWS_SomeIpXf_00163] ⌈ The Return Code field shall be 8 bit long. ⌋(*SRS_Xfrm_00008*)

[SWS_SomeIpXf_00164] ⌈ The Return Code field shall be used to signal whether a request has been successfully processed. ⌋(*SRS_Xfrm_00008*)

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] ⌈ Messages of Type REQUEST, REQUEST_NO_RETURN, and Notification have to set the Return Code to 0x00 (E_OK). ⌋(*SRS_Xfrm_00008*)

[SWS_SomeIpXf_00168] ⌈ The allowed Return Codes for specific message types shall be:

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].

⌋(*SRS_Xfrm_00008*)

7.2.3.8 Payload [variable size]

[SWS_SomeIpXf_00165] ⌈ The Payload field shall have variable size. ⌋(*SRS_Xfrm_00008*)

[SWS_SomeIpXf_00166] ⌈ The Payload field shall contain the transported data. ⌋(*SRS_Xfrm_00008*)

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

7.2.4 Serialization of Parameters and Data Structures

[SWS_SomeIpXf_00034] ⌈ The serialization shall be based on the `SenderReceiverInterface` or `ClientServerInterface` of the data. ⌋(*SRS_Xfrm_00101*)

[SWS_SomeIpXf_00169] ⌈ To allow migration the deserialization shall ignore parameters attached to the end of previously known parameter list. ⌋(*SRS_Xfrm_00101*)

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_00035] ⌈ The payload shall be aligned according to alignment of SOMEIPTransformationDescription which contains the memory alignment in Bits. For simplification the alignment should be a multiple of 8 Bit. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00037] ⌈ Alignment is always calculated from start of SOME/IP message. ⌋*(SRS_Xfrm_00101)*

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.

[SWS_SomeIpXf_00016] ⌈ 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. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00017] ⌈ If less data than expected are handed over to the SOME/IP transformer during deserialization of data, the following shall happen:

- if for the corresponding ISignal an initial value is specified (in serialized form) use that value to fill the missing elements.
- if no initial value is available abort deserialization with E_SER_MALFORMED_MESSAGE.

⌋*(SRS_Xfrm_00101)*

In the following the serialization of different parameters is specified.

7.2.4.1 Basic Datatypes

[SWS_SomeIpXf_00036] ⌈ The following basic datatypes shall be supported:

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)

⌋(*SRS_Xfrm_00101*)

The Byte Order is specified common for all parameters by `byteOrder` of `SOMEIP-TransformationDescription`. See chapter 7.2.2.

7.2.4.2 Structured Datatypes (structs)

[SWS_SomeIpXf_00042] ⌈ A struct shall be serialized in order of depth-first traversal. ⌋(*SRS_Xfrm_00101*)

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 an uint8 and an 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] ⌈ If attribute `sizeOfStructLengthFields` of `SOMEIP-TransformationISignalProps` is set to a value greater 0, a length field shall be inserted in front of every serialized struct. ⌋(*SRS_Xfrm_00101*)

Note:
This also applies to nested structs which means that additionally every nested struct has its own length field.

[SWS_SomeIpXf_00217] ⌈ The data type of the length field for a struct shall be determined by the value of `SOMEIPTransformationISignalProps.sizeOf-StructLengthFields` of the serialized `ISignal`:

- *uint8* if `sizeOfStructLengthFields` equals 1

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

⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00218] ⌈ 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. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00219] ⌈ 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. ⌋*(SRS_Xfrm_00101)*

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

Please note that the SOME/IP transformer only supports SOME/IP messages where all length fields of structs have the same size.

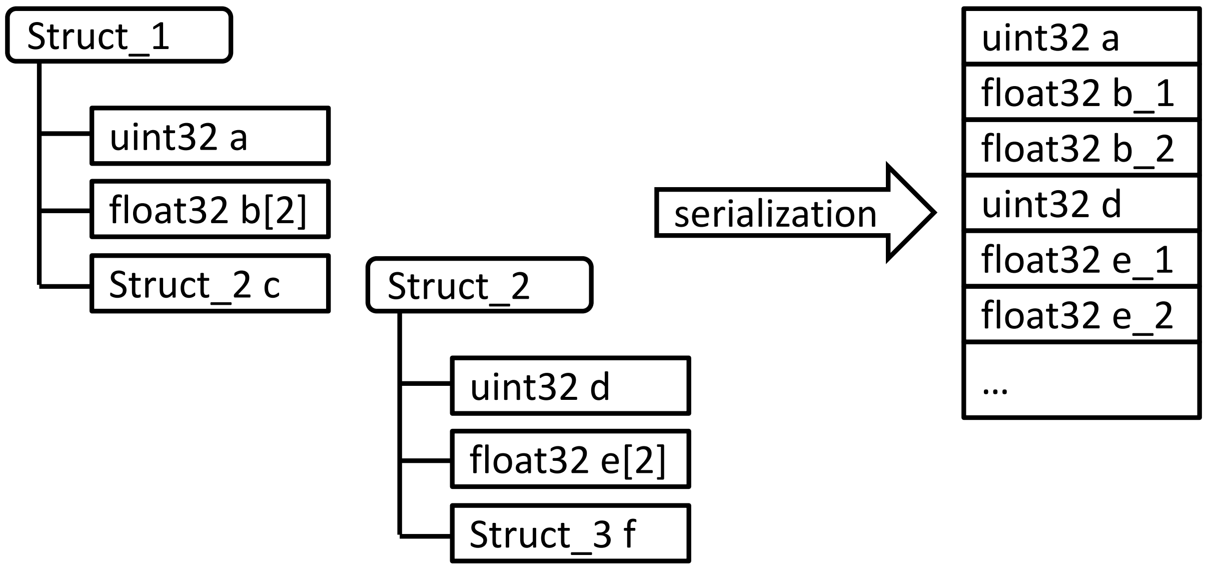

Figure 7.4: Serialization of Structs without Length Fields (Example)

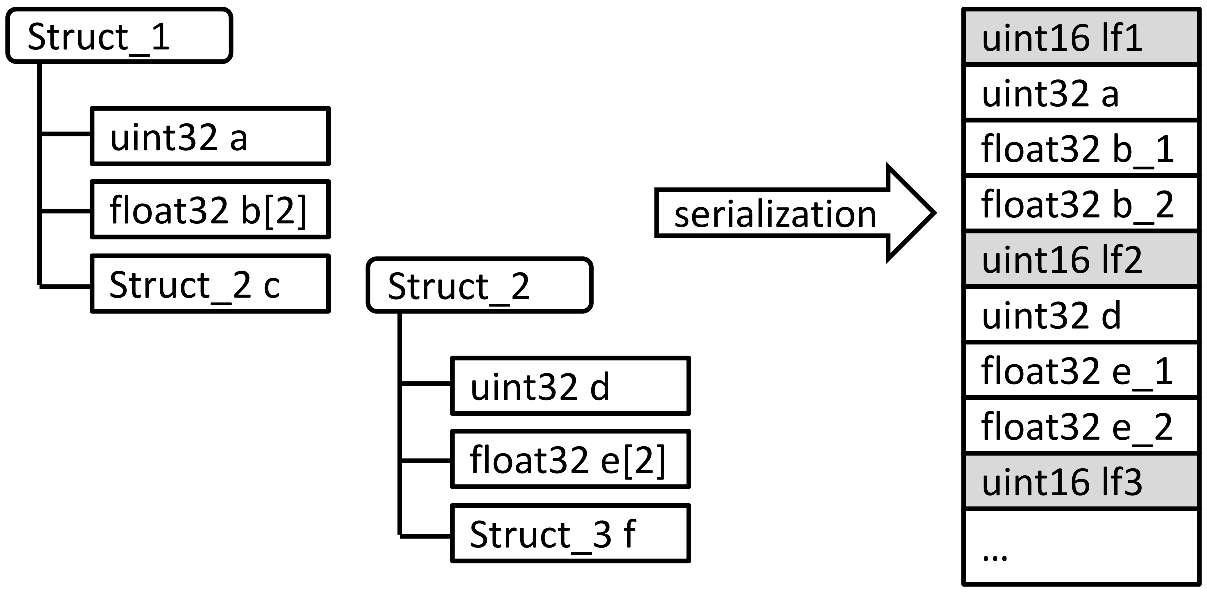

Figure 7.5: Serialization of Structs with Length Fields (Example)

7.2.4.3 Strings (fixed length)

[SWS_SomeIpXf_00053] ⌈ Strings shall be encoded using Unicode and terminated with a "\0"-character despite having a fixed length. Unused space shall be filled using "\0". ⌋*(SRS_Xfrm_00101)*

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

[SWS_SomeIpXf_00054] ⌈ Different Unicode encoding shall be supported including UTF-8, UTF-16BE, and UTF-16LE. Since these encoding have a dynamic length of bytes per character, the maximum length in bytes is up to three times the length of characters in UTF-8 plus 1 Byte for the termination with a "\0" or two times the length of the characters in UTF-16 plus 2 Bytes for a "\0". UTF-8 character can be up to 6 bytes and an UTF-16 character can be up to 4 bytes. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00055] ⌈ UTF-16LE and UTF-16BE strings shall be zero terminated with a "\0" character. This means they shall end with (at least) two 0x00 Bytes. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00056] ⌈ UTF-16LE and UTF-16BE strings shall have an even length. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00057] ⌈ For UTF-16LE and UTF-16BE strings having an odd length the last byte shall be ignored. ⌋*(SRS_Xfrm_00101)*

After removal of the last byte, the two bytes before shall be 0x00 bytes (termination) for a string to be valid.

[SWS_SomeIpXf_00058] ⌈ 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. ⌋ *(SRS_Xfrm_00101)*

For the specification of BOM, see [7] and [8].

[SWS_SomeIpXf_00059] ⌈ The receiving SOME/IP implementation shall check the BOM and handle this as an error. ⌋ *(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00060] ⌈ The BOM shall be added by the SOME/IP transformer. ⌋ *(SRS_Xfrm_00101)*

7.2.4.4 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.

7.2.4.5 Arrays (fixed length)

[SWS_SomeIpXf_00069] ⌈ The length of fixed length arrays is defined by the datatype definition. ⌋ *(SRS_Xfrm_00101)*

They can be seen as repeated elements. In chapter 7.2.4.7 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] ⌈ If attribute `sizeOfArrayLengthFields` of `SOMEIP-TransformationISignalProps` is set to a value greater 0, a length field shall be inserted in front of every serialized array. ⌋ *(SRS_Xfrm_00101)*

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

[SWS_SomeIpXf_00221] ⌈ 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

⌋ *(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00222] ⌈ 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. ⌋ *(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00223] ⌈ 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. ⌋ *(SRS_Xfrm_00101)*

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

Please note that the SOME/IP transformer only supports SOME/IP messages where all length fields of arrays have the same size.

7.2.4.5.1 One-dimensional

The one-dimensional arrays with fixed length n carry exactly n elements of the same type. The layout is shown in Figure 7.6.

[SWS_SomeIpXf_00070] ⌈ A one-dimensional array with fixed length shall be serialized by concatenating the array elements in order. ⌋ *(SRS_Xfrm_00101)*

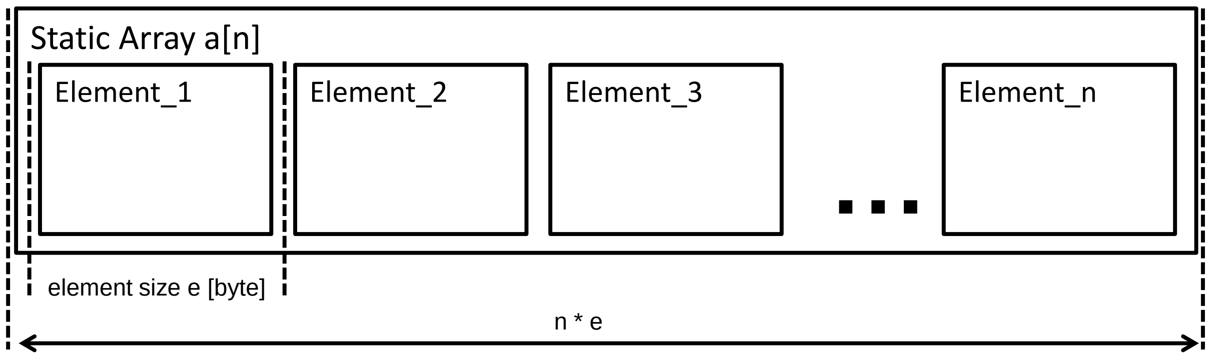

Figure 7.6: One-dimensional array (fixed length)

7.2.4.5.2 Multidimensional

[SWS_SomeIpXf_00072] ⌈ The serialization of multidimensional arrays shall happen in row-major order(in-memory layout of multidimensional arrays in the C++ programming language) ⌋ *(SRS_Xfrm_00101)*

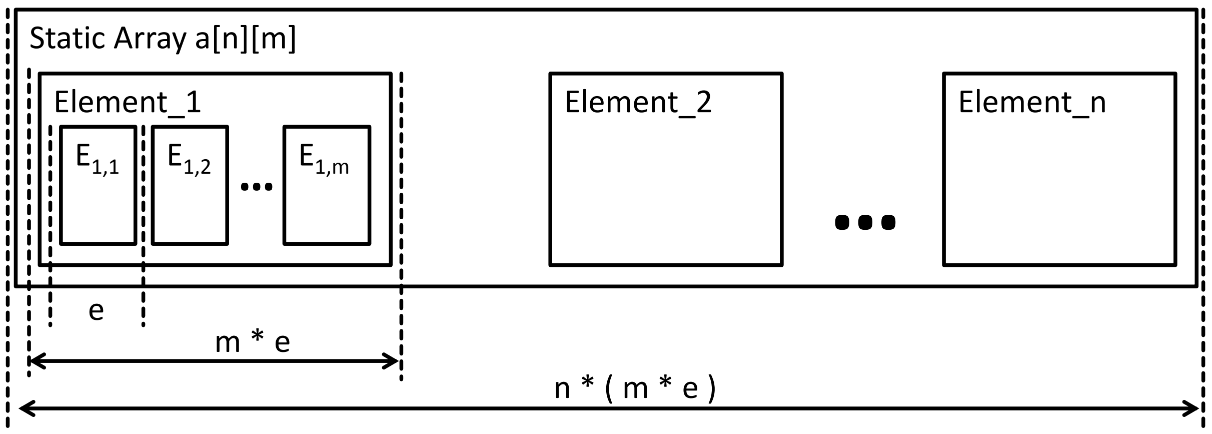

Figure 7.7: Multidimensional array (fixed length)

Consult AUTOSAR SWS RTE chapter 5.3.4.4 for Arrays.

7.2.4.6 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 7.2.4.7.

7.2.4.7 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] ([9, System Template]), [TPS_SWCT_01644], [TPS_SWCT_01645], [TPS_SWCT_01642] and [TPS_SWCT_01643].

[SWS_SomeIpXf_00076] ⌈ 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 length indicator shall be of data type uint8, uint16 or uint32. It shall be the smallest size which is still able to carry the maximum length of the following array.
- 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.

⌋*(SRS_Xfrm_00101)*

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 7.8 and Figure 7.9 where `L_1` and `L_2` denote the length in bytes.

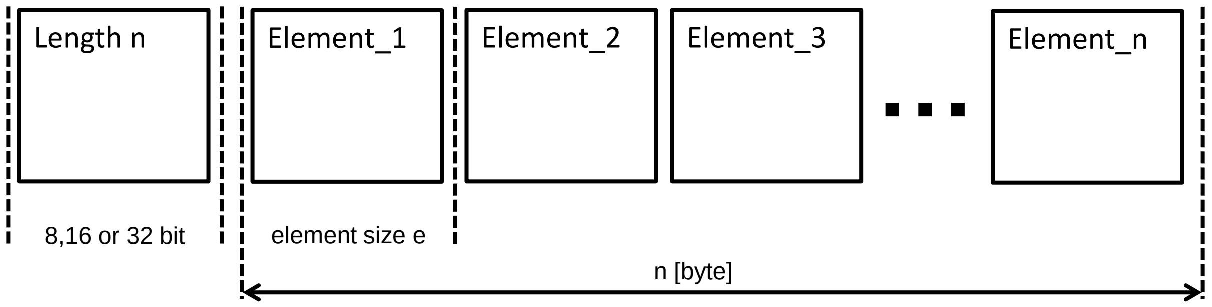

Figure 7.8: One-dimensional array (dynamic length) (Example)

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

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.

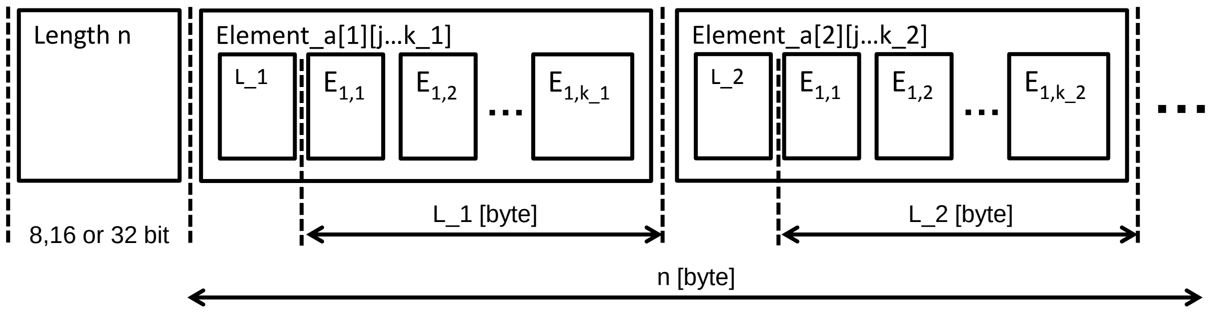

Figure 7.9: Multidimensional array (dynamic length) (Example)

In multidimensional arrays multiple length fields are needed.

It is even supported to have different length columns and different length rows in the same dimension. See `k_1` and `k_2` in Figure 7.9.

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

7.2.4.8 Bitfield

[SWS_SomeIpXf_00300] ⌈ Bitfields shall be transported as basic datatypes uint8/uint16/uint32. ⌋*()*

7.2.4.9 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.

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

[SWS_SomeIpXf_00088] ⌈ The default serialization layout of unions in SOME/IP is as follows:

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

⌋*(SRS_Xfrm_00101)*

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] ⌈ If attribute `sizeOfUnionLengthFields` of `SOMEIP-TransformationISignalProps` is set to a value greater 0, a length field shall be inserted in front of every serialized union. ⌋*(SRS_Xfrm_00101)*

Note:
This also applies to nested unions which means that additionally every nested union has its own length field.

[SWS_SomeIpXf_00225] ⌈ The data type of the length field for a 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

⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00226] ⌈ 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. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00227] ⌈ 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. ⌋*(SRS_Xfrm_00101)*

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

Please note that the SOME/IP transformer only supports SOME/IP messages where all length fields of unions have the same size.

The length of the type field shall be 32, 16, 8 or 0 bits. It shall be chosen as small as possible but shall be able to identify all different types.

The type field describes the type of the element.

[SWS_SomeIpXf_00098] ⌈ 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. ⌋*(SRS_Xfrm_00101)*

[SWS_SomeIpXf_00099] ⌈ 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]. ⌋*(SRS_Xfrm_00101)*

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

7.2.4.9.1 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

7.2.4.10 Example Map / Dictionary

Maps or dictionaries can be easily described as an array of key-value-pairs. The most basic way to implement a map or dictionary would be an array of a struct with two fields: key and value. Since the struct has no length field, this is as efficient as a special map or dictionary type could be. When choosing key and value as uint16, a serialized map with 3 entries looks like this:

Length = 12 Bytes	
key0	value0
key1	value1
key2	value2

7.3 Protocol specification

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

[SWS_SomeIpXf_00105] ⌈ The receiving SOME/IP implementation shall be able to receive unaligned SOME/IP messages. ⌋*(SRS_Xfrm_00008)*

7.3.1 Client/Server Communication

[SWS_SomeIpXf_00106] ⌈ 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

⌋*(SRS_Xfrm_00102)*

[SWS_SomeIpXf_00120] ⌈ To construct the payload 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`. ⌋*(SRS_Xfrm_00102)*

This can be seen graphically in Figure 7.10.

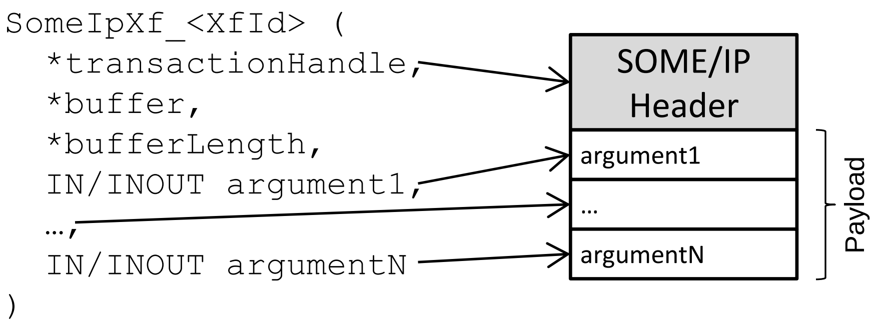

Figure 7.10: Example for serialization of a Client/Server Request

[SWS_SomeIpXf_00200] ⌈ 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], else it shall generate a normal response according to [SWS_SomeIpXf_00107]. ⌋*(SRS_Xfrm_00102)*

[SWS_SomeIpXf_00107] ⌈ 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)
- Place the return value of the executed `ClientServerOperation` into the Return Code field (see chapter 7.2.3.7) if the `ClientServerOperation` has at least one `PossibleError` defined.
  Use `E_OK` in the SOME/IP header if the `ClientServerOperation` has no `PossibleError` defined.

⌋*(SRS_Xfrm_00102)*

[SWS_SomeIpXf_00121] ⌈ To construct the payload 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 ClientServer-Operation. ⌋*(SRS_Xfrm_00102)*

This can be seen graphically in Figure 7.11.

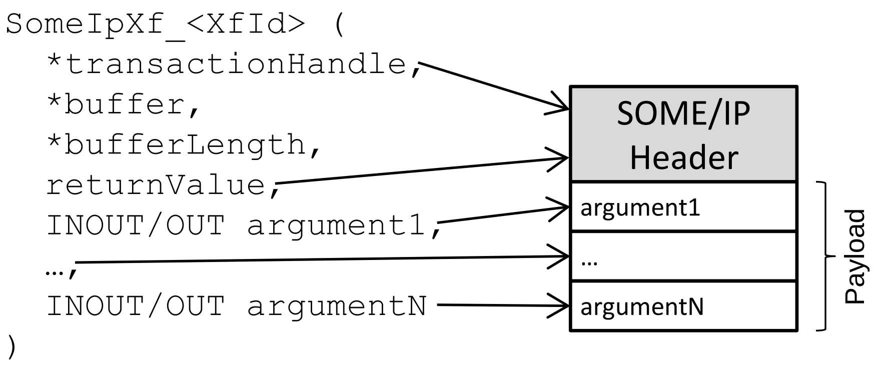

Figure 7.11: Example for serialization of a Client/Server Response

[SWS_SomeIpXf_00201] ⌈ 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 ERROR (i.e. 0x81)
- Adapt the return value by subtracting 0x80 from the parameter returnValue (calculation: $adaptedReturnValue = returnValue - 0x80$)
- Place the adaptedReturnValue into the Return Code field (see 7.2.3.7).

⌋*(SRS_Xfrm_00102)*

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] ⌈ 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]). ⌋*(SRS_Xfrm_00102)*

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.

7.3.2 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] ⌈ One Session Handling ID counter (32 Bit) has to be maintained per transformer function for Sender/Receiver communication (see [SWS_SomeIpXf_00138]) if `sessionHandlingSR` is set to `sessionHandlingActive`. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00213] ⌈ All Session Handling ID counters shall be initialized with 0x0001. ⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00108] ⌈ 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 `sessionHandlingSR` of `SOMEIPTransformationISignalProps` is not set to `sessionHandlingActive`
  - 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

⌋*(SRS_Xfrm_00102)*

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 behaviour of SOME/IP is for sender/receiver communication to ignore session IDs on receiver side.

[SWS_SomeIpXf_00176] ⌈ The payload of a message for Sender/Receiver communication shall consists of the serialized data element that is transported. ⌋*(SRS_Xfrm_00102)*

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

7.3.3 Unqueued External Trigger Events

[SWS_SomeIpXf_00204] ⌈ The SOME/IP transformer on the trigger source side of transformed external trigger events shall construct header and payload in the following way:

- Construct the payload
- Set the Request ID
  - to 0x00 if `sessionHandlingSR` of `SOMEIPTransformationISignalProps` is not set to `sessionHandlingActive`
  - 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

⌋*(SRS_Xfrm_00102)*

[SWS_SomeIpXf_00205] ⌈ The payload of a message for unqueued external trigger event communication shall be empty. ⌋*(SRS_Xfrm_00102)*

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

7.3.4 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] ⌈ The SOME/IP transformer shall use the Return Code error handling. ⌋*(SRS_Xfrm_00102, SRS_Xfrm_00103)*

Exceptions are specified in SOME/IP but not yet supported by this version of the SOME/IP transformer.

This can be used to handle all different application errors that might occur in the server. In addition, problems with the communication medium or intermediate components (e.g. switches) may occur, which have to be handled e.g. by means of reliable transport.

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 7.2.3.6). For more detailed errors the layout of the Error Message (Message Type 0x81) can carry specific fields for error handling, e.g. an Exception String. Error Messages are sent instead of Response Messages.

7.3.4.1 Return Code

[SWS_SomeIpXf_00112] ⌈ The Error Handling via Return Type shall be based on the `Std_ReturnType`. ⌋*(SRS_Xfrm_00102)*

[SWS_SomeIpXf_00113] ⌈ The Return Codes shall only be used for Client/Server communication ⌋*(SRS_Xfrm_00102)*

[SWS_SomeIpXf_00170] ⌈ In case of Client/Server communication the Return Code shall transport the `ApplicationError`s of the executed `ClientServerOperation` if no SOME/IP error occurred. ⌋*(SRS_Xfrm_00102)*

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] ⌈ The following Return Codes are currently defined and shall be implemented as described:

ID	Name	Description
0x00	E_OK	No error occurred
0x01	E_NOT_OK	An unspecified error occurred
0x02	SOMEIPXF_E_UNKNOWN_SERVICE	The requested Service ID is unknown.

0x03	SOMEIPXF_E_UNKNOWN_METHOD	The requested Method ID is unknown. Service ID is known.
0x04	SOMEIPXF_E_NOT_READY	Service ID and Method ID are known. Application not running.
0x05	SOMEIPXF_E_NOT_REACHABLE	System running the service is not reachable (internal error code only).
0x06	SOMEIPXF_E_TIMEOUT	A timeout occurred (internal error code only).
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. REQUEST_NO_RETURN for a method defined as REQUEST.)
0x0b - 0x1f	RESERVED	Reserved for generic SOME/IP errors. These errors will be specified in future versions of this document.
0x20 - 0x5e	-	Specific `ApplicationError`s of `ClientServerOperation`s. These errors are the application errors specified by the `ClientServerInterface`. As the range of `ApplicationError`s of the `ClientServerInterface` is `0x01-0x3F`, the value of an `ApplicationError` has to be adapted for transport over SOME/IP by adding `0x1F`.

⌋(*SRS_Xfrm_00102*)

7.3.4.2 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 7.12 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.

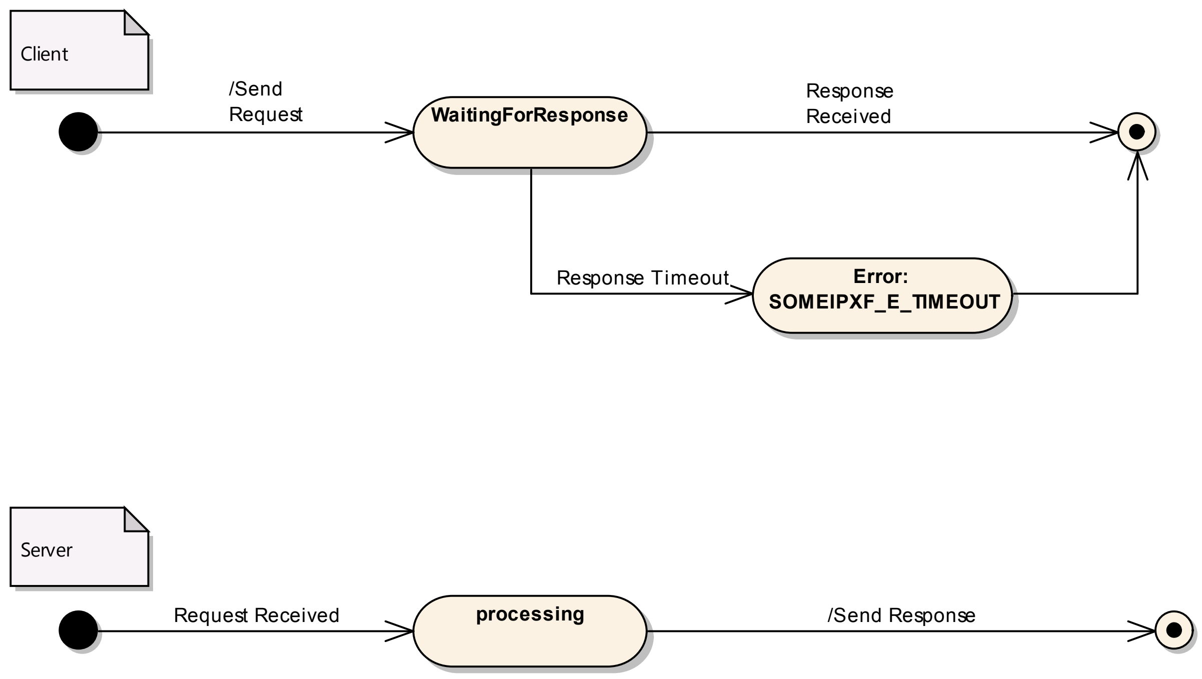

Figure 7.12: State Machines for Reliability "Maybe"

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 7.3.4.2.1 describes such optional reliability mechanisms.

7.3.4.2.1 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 7.13 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.

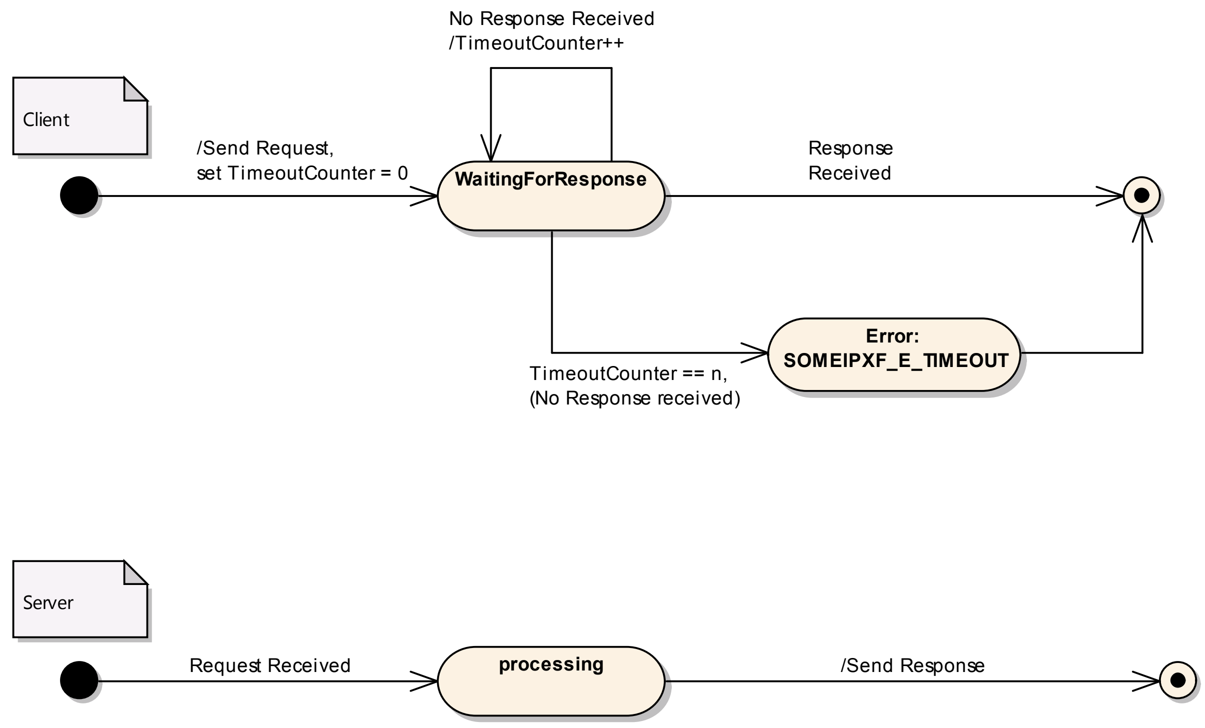

Figure 7.13: State Machines for Reliability "At least once" (idempotent operations)

7.4 Reserved and special identifiers for SOME/IP and SOME/IP-SD.

In this chapter an overview of reserved and special identifiers are shown.

[SWS_SomeIpXf_00130] ⌈ Reserved and special Service-IDs:

Service-ID	Description
0x0000	Reserved
0xFF00 - 0xFF1F	Reserved for Testing at OEM
0xFF20 - 0xFF3F	Reserved for Testing at Tier-1
0xFF40 - 0xFF5F	0xFF5F Reserved for ECU Internal Communication (Tier-1 proprietary)
0xFFFE	Reserved for announcing non-SOME/IP service instances.
0xFFFF	SOME/IP and SOME/IP-SD special service.

⌋*(SRS_Xfrm_00008)*

[SWS_SomeIpXf_00131] ⌈ Reserved and special Instance-IDs:

Instance-ID	Description
0x0000	Reserved
0xFFFF	All Instances

⌋(*SRS_Xfrm_00008*)

[SWS_SomeIpXf_00132] ⌈ Reserved and special Method-IDs/Event-IDs:

Method-ID	Description
0x0000	Reserved
0x7FFF	Reserved
0x8000	Reserved
0xFFFF	Reserved

⌋(*SRS_Xfrm_00008*)

[SWS_SomeIpXf_00133] ⌈ Method-IDs and Event-IDs of Service 0xFFFF:

Method-ID/Event-ID	Description
0x0000	SOME/IP Magic Cookie Messages
0x8000	SOME/IP Magic Cookie Messages
0x8100	SOME/IP-SD messages (events)

⌋(*SRS_Xfrm_00008*)

[SWS_SomeIpXf_00134] ⌈ Besides "otherserv" other names are supported by the configuration option. The following list gives an overview of the reserved names:

Name	Description
hostname	Used to name a host or ECU.
instancename	Used to name an instance of a service.
servicename	Used to name a service.
otherserv	Used for non-SOME/IP Services.

⌋(*SRS_Xfrm_00008*)

7.5 Development Errors

[SWS_SomeIPxf_00184] ⌈

Type of error	Related error code	Value
Error code if any other API service, except `GetVersionInfo` 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
API service called with wrong parameter	`SOMEIPXF_E_PARAM`	0x03
API service called with invalid pointer	`SOMEIPXF_E_PARAM_POINTER`	0x04

⌋(*SRS_BSW_00337*)

7.6 Production Errors

No production errors are specified for transformers.

7.7 Extended Production Errors

All Extended Production Errors valid for SOME/IP Transformer are specified in [3, ASWS Transformer General].

7.8 Error Notification

Defined in [10, SWS BSW General].

8 API specification

8.1 Imported types

There are no imported types from other modules beyond those specified in [3, ASWS 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.

8.2 Type definitions

[SWS_SomeIpXf_00183] ⌈

Name:	`SomeIpXf_ConfigType`		
Type:	`Structure`		
Element:		`implementation specific`	–
Description:	This is the type of the data structure containing the initialization data for the transformer.		

Table 8.1: SomeIpXf_ConfigType

⌋*(SRS_BSW_00404, SRS_BSW_00441)*

8.3 Function definitions

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

[SWS_SomeIpXf_00150] ⌈ 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 `transformer` 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_<transformerId>

[SWS_SomeIpXf_00138] ⌈

Service name:	SomeIpXf_<transformerId>	

Syntax:	`uint8 SomeIpXf_<transformerId>(`
	`uint8* buffer,`
	`uint16* bufferLength,`
	`const <type>* dataElement`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	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 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 an uint8 array containing the serialized data. 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.	

Table 8.2: SomeIpXf_transformerId1

where

- `type` is data type of the data element
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, ASWS 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] ⌈ The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00138] shall exist for the first reference in the list of ordered references `transformer` 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`, a `SenderRecRecordElementMapping` or a `SenderRecArrayElementMapping`. ⌋*()*

[SWS_SomeIpXf_00140] ⌈ 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] ⌈ 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 sessionHandlingSR is set to sessionHandlingActive. ⌋*()*

[SWS_SomeIpXf_00215] ⌈ When the Session Handling ID counter assigned to <transformerId> is 0xFFFF and gets incremented, it shall roll-over to 0x0001 (instead of 0x0000) if sessionHandlingSR is set to sessionHandlingActive. ⌋*()*

[SWS_SomeIpXf_00141] ⌈

Service name:	SomeIpXf_<transformerId>	

Syntax:	`uint8 SomeIpXf_<transformerId>(`
	`const Rte_Cs_TransactionHandleType* TransactionHandle,`
	`uint8* buffer,`
	`uint16* bufferLength,`
	`[Std_ReturnType returnValue,]`
	`<type> data_1, ...`
	`<type> data_n`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TransactionHandle	Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests.
	returnValue	Return value of the server runnable which needs to be serialized on server side for transmission to the calling client. This argument is only available for serializers of the response of a Client/Server communication and if the ClientServerOperation has at least one PossibleError defined.
	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 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 an uint8 array containing the serialized data. 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.

Table 8.3: SomeIpXf_transformerId2

where

- `type` is data type of the data element
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, ASWS Transformer General]).

⌋*()*

For the arguments of `ClientServerOperation` which are handed over to the transformer as `data_1`, ..., `data_n` the requirements to API parameters stated in chapter *API Parameters* of [5, SWS RTE] are valid (especially [SWS_Rte_01017], [SWS_Rte_01018] and [SWS_Rte_05107]).

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 correpsond 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 correpsond 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] ⌈ The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00141] shall exist for the first reference in the list of ordered references `transformer` 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] ⌈ 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] ⌈ 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. ⌋*(SRS_Xfrm_00105)*

[SWS_SomeIpXf_00206] ⌈

Service name:	SomeIpXf_<transformerId>	

Syntax:	`uint8 SomeIpXf_<transformerId>(`
	`uint8* buffer,`
	`uint16* 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 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 an uint8 array. 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.	

Table 8.4: SomeIpXf_transformerId3

where

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

⌋*(SRS_Xfrm_00002)*

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] ⌈ The function `SomeIpXf_<transformerId>` specified in [SWS_SomeIpXf_00206] shall exist for the first referenced `Transformation-Technology` in the ordered `transformerChain` of a `DataTransformation` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransfor-`

mation where the ISignal references a SystemSignal which is referenced by a TriggerToSignalMapping. ⌋(*SRS_Xfrm_00002*)

[SWS_SomeIpXf_00208] ⌈ 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. ⌋(*SRS_Xfrm_00002*)

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.

8.3.2 SomeIpXf_Inv_<transformerId>

[SWS_SomeIpXf_00144] ⌈

Service name:	SomeIpXf_Inv_<transformerId>	

Syntax:	`uint8 SomeIpXf_Inv_<transformerId>(`
	`const uint8* buffer,`
	`uint16 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):	None	
Parameters (out):	dataElement	Data element which is the result of the transformation and contains the deserialized data element
Return value:	uint8	0x00 (E_OK): Deserialization successful 0x81 (E_SER_GENERIC_ERROR): A generic error occurred 0x87 (E_SER_WRONG_PROTOCOL_VERSION): The version of the receiving transformer didn't match the sending transformer. 0x88 (E_SER_WRONG_INTERFACE_VERSION): Interface version of serialized data is not supported. 0x89 (E_SER_MALFORMED_MESSAGE): The received message is malformed. The transformer is not able to produce an output. 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.	

Table 8.5: SomeIpXf_Inv_transformerId1

where

- `type` is data type of the data element
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, ASWS 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] ⌈ The function `SomeIpXf_Inv_<transformerId>` specified in [SWS_SomeIpXf_00144] shall exist for the first reference in the list of ordered references `transformer` 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`, a `SenderRecRecordElementMapping` or a `SenderRecArrayElementMapping`. ⌋*()*

[SWS_SomeIpXf_00147] ⌈ 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_00145] ⌈

Service name:	SomeIpXf_Inv_<transformerId>	

Syntax:	`uint8 SomeIpXf_Inv_<transformerId>(`
	`Rte_Cs_TransactionHandleType* TransactionHandle,`
	`const uint8* buffer,`
	`uint16 bufferLength,`
	`[Std_ReturnType* returnValue,]`
	`[<type>* data_1,] ...`
	`[<type>* data_n]`
	`)`	

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):	TransactionHandle	Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests.
	returnValue	Return value of the server runnable which needs to be serialized on server side for transmission to the calling client. This argument is only available for deserializers of the response of a Client/Server communication and if the ClientServerOperation has at least one PossibleError defined.
	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)
Return value:	uint8	0x00 (E_OK): Deserialization successful 0x81 (E_SER_GENERIC_ERROR): A generic error occurred 0x87 (E_SER_WRONG_PROTOCOL_VERSION): The version of the receiving transformer didn't match the sending transformer. 0x88 (E_SER_WRONG_INTERFACE_VERSION): Interface version of serialized data is not supported. 0x89 (E_SER_MALFORMED_MESSAGE): The received message is malformed. The transformer is not able to produce an output. 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.	

Table 8.6: SomeIpXf_Inv_transformerId2

where

- `type` is data type of the data element
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, ASWS Transformer General]).

⌋*()*

For the arguments of `ClientServerOperation` which are handed over to the transformer as `data_1`, ..., `data_n` the requirements to API parameters stated in chapter *API Parameters* of [5, SWS RTE] are valid (especially [SWS_Rte_01019], [SWS_Rte_07082] and [SWS_Rte_05108]).

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 correpsond 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 correpsond 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_00148] ⌈

The function SomeIpXf_Inv_<transformerId> specified in [SWS_SomeIpXf_00145] shall exist for the first reference in the list of ordered references transformer 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] ⌈ 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. ⌋*()*

[SWS_SomeIpXf_00209] ⌈

Service name:	SomeIpXf_Inv_<transformerId>	

Syntax:	`uint8 SomeIpXf_Inv_<transformerId>(`
	`const uint8* buffer,`
	`uint16 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 0x81 (E_SER_GENERIC_ERROR): A generic error occurred 0x87 (E_SER_WRONG_PROTOCOL_VERSION): The version of the receiving transformer didn't match the sending transformer. 0x88 (E_SER_WRONG_INTERFACE_VERSION): Interface version of serialized data is not supported. 0x89 (E_SER_MALFORMED_MESSAGE): The received message is malformed. The transformer is not able to produce an output. 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.	

Table 8.7: SomeIpXf_Inv_transformerId3

where

- transformerId is the name pattern for the transformer specified in [SWS_Xfrm_00062] ([3, ASWS Transformer General]).

⌋(*SRS_Xfrm_00002*)

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] ⌈ 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. ⌋(*SRS_Xfrm_00002*)

[SWS_SomeIpXf_00211] ⌈ 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. ⌋(*SRS_Xfrm_00002*)

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_Init

[SWS_SomeIpXf_00181] ⌈

Service name:	SomeIpXf_Init	

Syntax:	`void SomeIpXf_Init(`
	`const SomeIpXf_ConfigType* config`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	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.	

Table 8.8: SomeIpXf_Init

⌋(*SRS_BSW_00407, SRS_BSW_00411*)

8.3.4 SomeIpXf_DeInit

[SWS_SomeIpXf_00182] ⌈

Service name:	SomeIpXf_DeInit

Syntax:	`void SomeIpXf_DeInit(`
	`void`
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service deinitializes the transformer.

Table 8.9: SomeIpXf_DeInit

⌋(*SRS_BSW_00407*, *SRS_BSW_00411*)

8.3.5 SomeIpXf_GetVersionInfo

[SWS_SomeIpXf_00180] ⌈

Service name:	SomeIpXf_GetVersionInfo	

Syntax:	`void SomeIpXf_GetVersionInfo(`
	`Std_VersionInfoType* VersionInfo`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	VersionInfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of the called transformer module.	

Table 8.10: SomeIpXf_GetVersionInfo

⌋(*SRS_BSW_00407*, *SRS_BSW_00411*)

8.4 Callback notifications

There are no callback notifications.

8.5 Scheduled functions

SOME/IP Transformer has no scheduled functions

8.6 Expected interfaces

There are no expected interfaces.

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, ASWS Transformer General] shall be used.

[SWS_SomeIpXf_00185] ⌈ The `apiServicePrefix` of the SOME/IP transformer's EcuC shall be set to `SomeIpXf`. ⌋*(SRS_BSW_00159)*

A 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	**ApplicationError**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
errorCode	Integer	1	attr	The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification).

Table A.1: ApplicationError

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,Data Prototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
direction	ArgumentDirecti onEnum	1	attr	This attribute specifies the direction of the argument prototype.
serverArgu mentImplP olicy	ServerArgument ImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures and to the value useArrayBaseType for arrays.
typeBluepri nt	AutosarDataTyp e	0..1	ref	This allows to denote the intended type within blueprints. It shall be replaced by a proper type when deriving Interfaces from the Blueprint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivation Time

Table A.2: ArgumentDataPrototype

Class	**ClientServerInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	1..*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table A.3: ClientServerInterface

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

Table A.4: ClientServerOperation

Class	**ClientServerToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	This element maps the ClientServerOperation to call- and return-SystemSignals. The serialization is defined by the referenced SerializationTechnology. **Tags:** atp.Status=draft			
Base	ARObject,DataMapping			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
callSignal	SystemSignal	1	ref	Reference to the callSignal to which the IN and INOUT ArgumentDataPrototypes are mapped.
clientServerOperation	ClientServerOperation	1	iref	Reference to a ClientServerOperation, which is mapped to a call SystemSignal and a return SystemSignal.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
lengthClientId	PositiveInteger	0..1	attr	This attribute defines the length of the used client identifier in bits. If the attribute does not exist or its value is set to 0 this means that the client identifier is not used. Please note that this attribute is deprecated and will be removed in future (Value is fixed to UInt16). **Tags:** atp.Status=obsolete; atp.StatusRevisionBegin=4.2.2
lengthSequenceCounter	PositiveInteger	0..1	attr	The purpose of a sequence counter is to map a response to the correct request of a known client. This attribute describes the length of the used sequence counter in bits. If the attribute does not exist or its value is set to 0 this means that the sequence counter is not used. Please note that this attribute is deprecated and will be removed in future (Value is fixed to UInt16). **Tags:** atp.Status=obsolete; atp.StatusRevisionBegin=4.2.2
returnSignal	SystemSignal	0..1	ref	Reference to the returnSignal to which the OUT and INOUT ArgumentDataPrototypes are mapped. **Tags:** atp.Status=shallBecomeMandatory

Table A.5: ClientServerToSignalMapping

Class	**DataTransformation**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	A DataTransformation represents a transformer chain. It is an ordered list of transformers.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
executeDespiteDataUnavailability	Boolean	1	attr	Specifies whether the transformer is executed even if no input data are available.
transformerChain (ordered)	TransformationTechnology	1..*	ref	

Table A.6: DataTransformation

Enumeration	**DataTransformationErrorHandlingEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions

Note	This enumeration defines different ways how runnables shall handle transformer errors.
Literal	***Description***
noTrans-formerError Handling	A runnable does not handle transformer errors.
transformer ErrorHan-dling	The runnable implements the handling of transformer errors.

Table A.7: DataTransformationErrorHandlingEnum

Class	**EcucModuleDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used as the top-level element for configuration definition for Software Modules, including BSW and RTE as well as ECU Infrastructure. **Tags:** atp.recommendedPackage=EcucModuleDefs			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpDefinition,Collectable Element,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
apiService Prefix	CIdentifier	0..1	ref	For CDD modules this attribute holds the apiServicePrefix. The shortName of the module definition of a Complex Driver is always "Cdd". Therefore for CDD modules the module apiServicePrefix is described with this attribute.
container	EcucContainerD ef	1..*	aggr	Aggregates the top-level container definitions of this specific module definition. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName xml.sequenceOffset=11
postBuildV ariantSupp ort	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.
refinedMod uleDef	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. **Stereotypes:** atpUriDef

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
supported ConfigVari ant	EcucConfigurati onVariantEnum	*	attr	Specifies which ConfigurationVariants are supported by this software module. This attribute is optional if the EcucModuleDef 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.

Table A.8: EcucModuleDef

Class	**ISignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Signal of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal is sent in different SignalIPdus to multiple receivers. To support the RTE "signal fan-out" each SignalIPdu contains ISignals. If the same System Signal is to be mapped into several SignalIPdus there is one ISignal needed for each ISignalToIPduMapping. ISignals describe the Interface between the Precompile configured RTE and the potentially Postbuild configured Com Stack (see ECUC Parameter Mapping). In case of the SystemSignalGroup an ISignal must be created for each SystemSignal contained in the SystemSignalGroup. **Tags:** atp.recommendedPackage=ISignals			
Base	ARObject,CollectableElement,FibexElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataTransf ormation	DataTransforma tion	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignal. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataTransformation, variation Point.shortLabel vh.latestBindingTime=codeGenerationTime

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataTypePolicy	DataTypePolicyEnum	1	attr	With the aggregation of SwDataDefProps an ISignal specifies how it is represented on the network. This representation follows a particular policy. Note that this causes some redundancy which is intended and can be used to support flexible development methodology as well as subsequent integrity checks. If the policy "networkRepresentationFromComSpec" is chosen the network representation from the ComSpec that is aggregated by the PortPrototype shall be used. If the "override" policy is chosen the requirements specified in the PortInterface and in the ComSpec are not fulfilled by the networkRepresentationProps. In case the System Description doesn't use a complete Software Component Description (VFB View) the "legacy" policy can be chosen.
iSignalProps	ISignalProps	0..1	aggr	Additional optional ISignal properties that may be stored in different files. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=iSignalProps
initValue	ValueSpecification	0..1	aggr	Optional definition of a ISignal's initValue in case the System Description doesn't use a complete Software Component Description (VFB View). This supports the inclusion of legacy system signals. This value can be used to configure the Signal's "InitValue". If a full DataMapping exist for the SystemSignal this information may be available from a configured SenderComSpec and ReceiverComSpec. In this case the initvalues in SenderComSpec and/or ReceiverComSpec override this optional value specification. Further restrictions apply from the RTE specification.
length	Integer	1	attr	Size of the signal in bits. The size needs to be derived from the mapped VariableDataPrototype according to the mapping of primitive DataTypes to BaseTypes as used in the RTE. Indicates maximum size for dynamic length signals. The ISignal length of zero bits is allowed.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
networkRepresentationProps	SwDataDefProps	0..1	aggr	Specification of the actual network representation. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType. The optional baseType attributes "memAllignment" and "byteOrder" shall not be used. The attribute "dataTypePolicy" in the SystemTemplate element defines whether this network representation shall be ignored and the information shall be taken over from the network representation of the ComSpec. If "override" is chosen by the system integrator the network representation can violate against the requirements defined in the PortInterface and in the network representation of the ComSpec. In case that the System Description doesn't use a complete Software Component Description (VFB View) this element is used to configure "ComSignalDataInvalidValue" and the Data Semantics.
systemSignal	SystemSignal	1	ref	Reference to the System Signal that is supposed to be transmitted in the ISignal.
transformationISignalProps	TransformationISignalProps	*	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.

Table A.9: ISignal

Class	**Implementation (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
buildActionManifest	BuildActionManifest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	1..*	aggr	Specifies the provided implementation code.
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
generated Artifact	DependencyOn Artifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is «atpSplitable» because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcSupport
programmi ngLanguag e	Programmingla nguageEnum	1	attr	Programming language the implementation was created in.
requiredArt ifact	DependencyOn Artifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
requiredGe neratorToo l	DependencyOn Artifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
resourceC onsumptio n	ResourceConsu mption	1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
swVersion	RevisionLabelSt ring	1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swcBswMapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or BswImplementtion or for both.
usedCodeGenerator	String	0..1	attr	Optional: code generator used.
vendorId	PositiveInteger	1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table A.10: Implementation

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table A.11: ImplementationDataType

Class	**InternalBehavior (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
constantMemory	ParameterDataPrototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this InternalBehavior. The shortName of ParameterDataPrototype has to be equal to the "C' identifier of the described constant. The characteristic value(s) might be shared between SwComponentPrototypes of the same SwComponentType. 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. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=constantValueMapping
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **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. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
exclusiveAreaNestingOrder	ExclusiveAreaNestingOrder	*	aggr	This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurment variables implemented by this software component. 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. The shortName of the VariableDataPrototype has to be equal with the "C' identifier of the described variable. The aggregation of staticMemory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.12: InternalBehavior

Class	**PortAPIOption**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
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 RunnableEntity to the PortPrototype).			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
enableTakeAddress	Boolean	1	attr	If set to true, the software-component is able to use the API reference for deriving a pointer to an object.
errorHandling	DataTransformationErrorHandlingEnum	0..1	attr	This specifies whether the RunnableEntitys which access a PortPrototype that it referenced by this PortAPIOption shall specifically handle transformer errors or not.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
indirectAPI	Boolean	1	attr	If set to true this attribute specifies an "indirect API" to be generated for the associated port which means that the SWC 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	1	ref	The option is valid for generated functions related to communication over this port
portArgValue (ordered)	PortDefinedArgumentValue	*	aggr	An argument value defined by this port.

Table A.13: PortAPIOption

Class	**PortDefinedArgumentValue**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
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 ClientServerInterface.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	ValueSpecification	1	aggr	Specifies the actual value.
valueType	ImplementationDataType	1	tref	The implementation type of this argument value. It should not be composite type or a pointer. **Stereotypes:** isOfType

Table A.14: PortDefinedArgumentValue

Enumeration	**SOMEIPTransformerSessionHandlingEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer
Note	Enables or disable session handling for SOME/IP transformer
Literal	***Description***
sessionHandlingActive	The SOME/IP Transformer shall use session handling
sessionHandlingInactive	The SOME/IP Transformer doesn’t use session handling

Table A.15: SOMEIPTransformerSessionHandlingEnum

Class	**SenderRecArrayElementMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	The SenderRecArrayElement may be a primitive one or a composite one. If the element is primitive, it will be mapped to the SystemSignal (multiplicity 1). If the VariableDataPrototype that is referenced by SenderReceiverToSignalGroupMapping is typed by an ApplicationDataType the reference to the ApplicationArrayElement shall be used. If the VariableDataPrototype is typed by the ImplementationDataType the reference to the ImplementationArrayElement shall be used. If the element is composite, there will be no mapping to the SystemSignal (multiplicity 0). In this case the ArrayElementMapping element will aggregate the TypeMapping element. In that way also the composite datatypes can be mapped to SystemSignals. Regardless whether composite or primitive array element is mapped the indexed element always needs to be specified.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
complexTypeMapping	SenderRecCompositeTypeMapping	0..1	aggr	This aggregation will be used if the element is composite.
indexedArrayElement	IndexedArrayElement	1	aggr	Reference to an indexed array element in the context of the dataElement or in the context of a composite element.
systemSignal	SystemSignal	0..1	ref	Reference to the system signal used to carry the primitive ApplicationArrayElement.

Table A.16: SenderRecArrayElementMapping

Class	**SenderRecRecordElementMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of a primitive record element to a SystemSignal. If the VariableDataPrototype that is referenced by SenderReceiverToSignalGroupMapping is typed by an ApplicationDataType the reference applicationRecordElement shall be used. If the VariableDataPrototype is typed by the ImplementationDataType the reference implementationRecordElement shall be used. Either the implementationRecordElement or applicationRecordElement reference shall be used. If the element is composite, there will be no mapping to the SystemSignal (multiplicity 0). In this case the RecordElementMapping element will aggregate the complexTypeMapping element. In that way also the composite datatypes can be mapped to SystemSignals.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
applicationRecordElement	ApplicationRecordElement	0..1	ref	Reference to an ApplicationRecordElement in the context of the dataElement or in the context of a composite element. This reference shall only be used if the VariableDataPrototype that is referenced by the SenderReceiverToSignalGroupMapping.dataElement is typed by an ApplicationDataType.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
complexTypeMapping	SenderRecCompositeTypeMapping	0..1	aggr	This aggregation will be used if the element is composite.
implementationRecordElement	ImplementationDataTypeElement	0..1	ref	Reference to an ImplementationRecordElement in the context of the dataElement or in the context of a composite element. This reference shall only be used if VariableDataPrototype that is referenced by the SenderReceiverToSignalGroupMapping.dataElement is typed by an ImplementationDataType.
systemSignal	SystemSignal	0..1	ref	Reference to the system signal used to carry the primitive ApplicationRecordElement.

Table A.17: SenderRecRecordElementMapping

Class	**SenderReceiverInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,DataInterface,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1..*	aggr	The data elements of this SenderReceiverInterface.
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement

Table A.18: SenderReceiverInterface

Class	**SenderReceiverToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of a sender receiver communication data element with a primitive datatype to a signal.			
Base	ARObject,DataMapping			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1	iref	Reference to the data element, which ought to be sent over the Communication bus.
systemSignal	SystemSignal	1	ref	Reference to the system signal used to carry the data element.

Table A.19: SenderReceiverToSignalMapping

Class	**SystemSignal**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
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. **Tags:** atp.recommendedPackage=SystemSignals			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
dynamicLength	Boolean	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.

Table A.20: SystemSignal

Class	≪atpVariation≫ **TransformationISignalProps (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	TransformationISignalProps holds all the attributes for the different TransformationTechnologies that are ISignal specific. **Tags:** vh.latestBindingTime=postBuild			
Base	ARObject,Describable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
csErrorReaction	CSTransformerErrorReactionEnum	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.
transformer	TransformationTechnology	1	ref	Reference to the TransformationTechnology description that contains transformer specific and ISignal independent configuration properties.

Table A.21: TransformationISignalProps

Class	**TransformationTechnology**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	A TransformationTechnology is a transformer inside a transformer chain. **Tags:** xml.namePlural=TRANSFORMATION-TECHNOLOGIES			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
bufferProperties	BufferProperties	1	aggr	Aggregation of the mandatory BufferProperties.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
needsOriginalData	Boolean	0..1	attr	Specifies whether this transformer gets access to the SWC's original data.
protocol	String	1	attr	Specifies the protocol that is implemented by this transformer.
transformationDescription	TransformationDescription	0..1	aggr	A transformer can be configured with transformer specific parameters which are represented by the TransformerDescription. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild
transformerClass	TransformerClassEnum	1	attr	Specifies to which transformer class this transformer belongs.
version	String	1	attr	Version of the implemented protocol.

Table A.22: TransformationTechnology

Class	**TriggerToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
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			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
systemSignal	SystemSignal	1	ref	This is the SystemSignal taken to transport the Trigger over the network. **Tags:** xml.sequenceOffset=20
trigger	Trigger	1	iref	This represents the Trigger that shall be used to trigger RunnableEntities deployed to a remote ECU. **Tags:** xml.sequenceOffset=10

Table A.23: TriggerToSignalMapping

B 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)

C Examples

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

C.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
	1	E_DATA_INCONSISTENT
	2	E_UNKNOWN_ERROR

Table C.1: ClientServerInterface SomeCSInterface

Operations

Name	SomeCSOperation

Comments	The ClientServerOperation which is used to demonstrate how the SOME/IP serialization for Client/Sever communication works	
Variation	–	
Parameters	inputParam1	
	Comment	A parameter which is handed over from the Client to the Server
	Type	uint8
	Variation	–
	Direction	IN
	inputParam2	
	Comment	A parameter which is handed over from the Client to the Server
	Type	uint16
	Variation	–
	Direction	IN
	biDirectionalParam	
	Comment	A parameter which is handed over from the Client to the Server, modified by the Server and handed back to the Client
	Type	someStruct
	Variation	–
	Direction	INOUT
	outputParam1	
	Comment	A parameter which is handed over from the Server to the Client
	Type	uint16
	Variation	–
	Direction	OUT
	outputParam2	
	Comment	A parameter which is handed over from the Server to the Client
	Type	uint32
	Variation	–
	Direction	OUT
Possible Errors	E_OK	Operation successful
	E_DATA_ INCONSISTENT	Data are inconsistent
	E_UNKNOWN_ERROR	An unknown error occured

Table C.2: Operation SomeCSOperation

C.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:

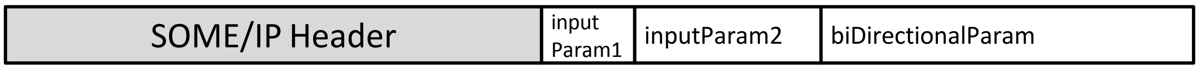

Figure C.1: Example for serialized data of the Client/Server Request

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)
```

C.1.2 Server

On the server side the ClientServerOperation is implemented by a runnable with the following signature which now contains the PortDefinedArgumentValues (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:

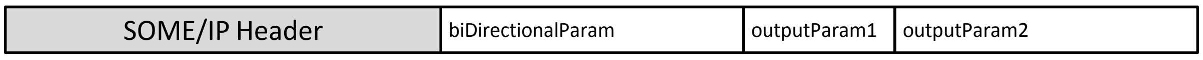

Figure C.2: Example for serialized data of the Client/Server Response

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