Document Title	General Specification of Transformers
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	658
Document Classification	Standard

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.0

Document Change History

Date	Release	Changed by	Description
2016-11-30	4.3.0	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2015-07-31	4.2.2	AUTOSAR Release Management	• Transformation of intra-ECU communication • Transformation of external-trigger events • Autonomous error responses of transformers • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2014-10-31	4.2.1	AUTOSAR Release Management	Initial Release

Disclaimer

Disclaimer

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

Transformer enable AUTOSAR systems to use a data transformation mechanism to linearize and transform data.

Transformers can be concatenated to transformer chains which are executed by the RTE for intra-ECU and inter-ECU communcation that is configured to be transformed.

A transformer provides well defined function signatures per each communication relation (port based and signal based), which is marked for transformation. The function signature depends on the transmitted data elements (Client/Server operation signature or Sender/Receiver interface signature) only. The output of a transformer will be always a linear byte array.

A more powerful system can chain multiple transformers where the input of the first transformer in the chain gets the data from the RTE. Each following transformer uses the output of the preceding transformer as input. All transformers following the first one then have generic signature with just a byte array as IN and OUT parameter. Such an architecture could be used to design systems, where you can flexibly add functionality like safety or security protection to a serialized stream.

2 Acronyms and Abbreviations

There are no acronyms and abbreviations relevant to this document that are not included in the [1, AUTOSAR glossary].

3 Related documentation

3.1 Input documents

Bibliography

[1] Glossary
AUTOSAR_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[3] System Template
AUTOSAR_TPS_SystemTemplate

[4] Specification of SOME/IP Transformer
AUTOSAR_SWS_SOMEIPTransformer

[5] Specification of RTE Software
AUTOSAR_SWS_RTE

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

3.2 Related standards and norms

Not applicable.

3.3 Related specification

Not applicable.

4 Constraints and assumptions

4.1 Limitations

Both data transformation and communication itself are very extensive fields and can get quite complex because a lot of use cases and scenarios are theoretically possible. Because these have a big impact on the functionality of transformer (especially in the RTE), this diversity makes it necessary to impose a few restrictions and assumptions to the transformers.

If the transformation targets primarily the serialization of large complex data elements,it is most efficient when the transformation is used for communication over busses with large PDU sizes (e.g. Ethernet). If busses with small PDU size are used (e.g CAN), the byte array produced by the serializer would have to be spanned over multiple PDUs which is possible but inefficient.

Subject to transformation are the data elements (VariableDataPrototypes) of ports typed with SenderReceiverInterfaces, the operations (ClientServerOperations) of ports typed with ClientServerInterfaces and non-queued external trigger events of ports typed with TriggerInterfaces with swImplPolicy not set to queued.

This imposes the majority of restrictions and is therefore the most important contraint! As a consequence of this decision, it is not possible to transform whole PDUs. The reason for this is the fact that inside the RTE (where the transformation happens) there exist no PDUs because these are built inside the Com module.

Nonetheless, it is still possible to aggregate multiple transformed data elements of Sender/Receiver-Communication into one large PDU inside Com (each transformed data element is visible within Com as an ISignal). But in this case, all data elements/ISignals contained in this PDU are transformed independently from each other, each including its own header (if the transformation adds headers). As a consequence of this, it is not possible to transform data structures where the data structure's sub-elements are produced by different data elements of different PPortPrototypes/SWCs.

The length of the transformer chains is not limited by the solutions chosen within this concept. But to enable a memory efficient configuration and implementation, the maximum length is artificially limited to 255 because current use cases see a maximum chain length of 3.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

There are not dependencies to AUTOSAR SWS modules.

5.1 File structure

5.1.1 Code file structure

The code file structure of transformers is defined by the [2, SWS BSW General] as all transformers are BSW modules. Deviations are specified in the SWS documents of the specific transformers.

5.1.2 Header file structure

The header file structure of transformers is defined by the [2, SWS BSW General] as all transformers are BSW modules. Deviations are specified in the SWS documents of the specific transformers.

6 Requirements Tracing

The following table references the SRS requirements which are fulfilled by this document.

Feature	Description	Satisfied by
[SRS_BSW_00337]	Classification of development errors	[SWS_Xfrm_00061]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Xfrm_00060]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Xfrm_00057] [SWS_Xfrm_00058] [SWS_Xfrm_00059]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Xfrm_00057] [SWS_Xfrm_00058] [SWS_Xfrm_00059]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_Xfrm_00060]
[SRS_BSW_00466]	Classification of extended production errors	[SWS_Xfrm_00070] [SWS_Xfrm_00071]
[SRS_BSW_00469]	Fault detection and healing of production errors and extended production errors	[SWS_Xfrm_00070] [SWS_Xfrm_00071]
[SRS_Xfrm_00001]	A transformer shall work on data given by the Rte	[SWS_Xfrm_00017] [SWS_Xfrm_00018] [SWS_Xfrm_00019] [SWS_Xfrm_00020] [SWS_Xfrm_00021] [SWS_Xfrm_00022] [SWS_Xfrm_00023] [SWS_Xfrm_00024] [SWS_Xfrm_00025] [SWS_Xfrm_00048] [SWS_Xfrm_CONSTR_09094] [SWS_Xfrm_CONSTR_09095] [SWS_Xfrm_CONSTR_09096]

[SRS_Xfrm_00002]	A transformer shall provide fixed interfaces	[SWS_Xfrm_00034] [SWS_Xfrm_00036] [SWS_Xfrm_00037] [SWS_Xfrm_00038] [SWS_Xfrm_00039] [SWS_Xfrm_00040] [SWS_Xfrm_00041] [SWS_Xfrm_00042] [SWS_Xfrm_00043] [SWS_Xfrm_00044] [SWS_Xfrm_00045] [SWS_Xfrm_00046] [SWS_Xfrm_00047] [SWS_Xfrm_00052] [SWS_Xfrm_00053] [SWS_Xfrm_00062] [SWS_Xfrm_00100] [SWS_Xfrm_00102] [SWS_Xfrm_00103] [SWS_Xfrm_00104] [SWS_Xfrm_00105] [SWS_Xfrm_00106] [SWS_Xfrm_00107]
[SRS_Xfrm_00003]	A Transformer shall support in-place and copy buffering	[SWS_Xfrm_00010] [SWS_Xfrm_00011] [SWS_Xfrm_00012] [SWS_Xfrm_00013] [SWS_Xfrm_00014]
[SRS_Xfrm_00004]	A transformer shall support error handling	[SWS_Xfrm_00026] [SWS_Xfrm_00027] [SWS_Xfrm_00028] [SWS_Xfrm_00029] [SWS_Xfrm_00030] [SWS_Xfrm_00051]
[SRS_Xfrm_00005]	A transformer shall be able to deal with more data than expected	[SWS_Xfrm_00008] [SWS_Xfrm_00049] [SWS_Xfrm_00108]
[SRS_Xfrm_00006]	A Transformer shall support concurrent execution	[SWS_Xfrm_00001] [SWS_Xfrm_00009] [SWS_Xfrm_00054] [SWS_Xfrm_00055] [SWS_Xfrm_00056] [SWS_Xfrm_00101]
[SRS_Xfrm_00007]	A deserializer transformer shall support extraction of data	[SWS_Xfrm_00048]
[SRS_Xfrm_00008]	A transformer shall specify its output format	[SWS_Xfrm_00002] [SWS_Xfrm_00003] [SWS_Xfrm_00004] [SWS_Xfrm_00005] [SWS_Xfrm_00006] [SWS_Xfrm_00007]

[SRS_Xfrm_00010]	Each transformer class shall provide a fixed set of abstract errors	[SWS_Xfrm_00029] [SWS_Xfrm_00030] [SWS_Xfrm_00031] [SWS_Xfrm_00032] [SWS_Xfrm_00033] [SWS_Xfrm_00050]
[SRS_Xfrm_00011]	A transformer shall belong to a specific transformer class	[SWS_Xfrm_00030]

7 Functional Specification

A transformers takes data from the RTE, works on them and returns the output back to the RTE. It can both serialize/linearize data (transform them from a structured into a linear form) and transform (modify or extend linear data) them (e.g add a checksum).

Transformers are BSW modules in the Communication Service Cluster which provides communication services to the RTE. The transformers are executed by the RTE when the RTE needs the service which a transformer provides.

A transformer is no library because transformers can hold an internal state but they can work as well stateless.

[SWS_Xfrm_00001] ⌈ Transformers shall be stateful only, if the dedicated transformer functionality requires maintaining a transformer state. ⌋*(SRS_Xfrm_00006)*

Please note that stateful transformers cannot be used like a library.

It is possible to connect a set of transformers together into a transformer chain. The RTE coordinates the execution of the transformer chain and calls the transformers of the chain exactly in the specified order. Using that mechanism, intra-ECU and inter-ECU communcation is transformed if configured accordingly. This configuration is done in the [3, System Template]. The maximum length of a transformer chain is limited to 255 transformers.

The order of transformers configured in the [3, System Template] represents the order on the sending side. The order on the receiving side is the inverse of the sending side.

An example of inter-ECU data transformation is shown in Figure 7.1.

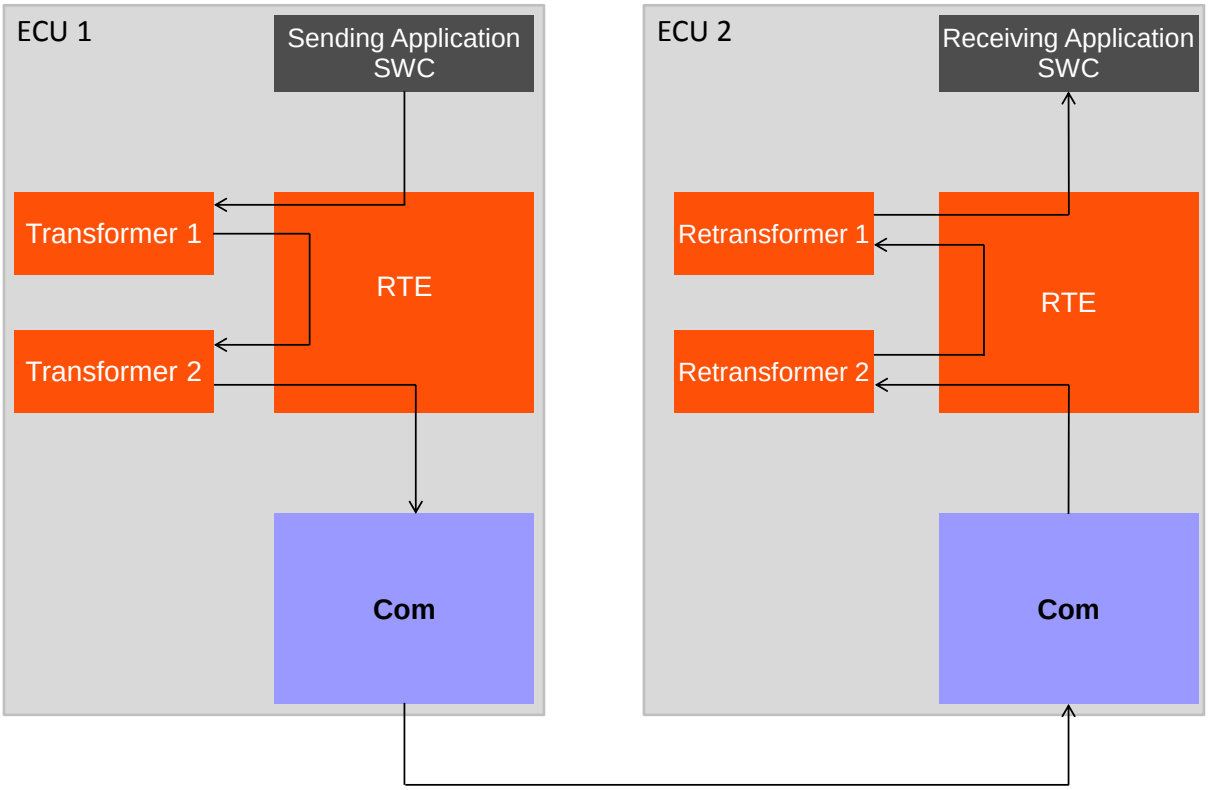

Figure 7.1: Transformer Example for Inter-ECU Communication

In this example, a SWC sends complex data which are transformed using a transformer chains with two transformers. Transformer 1 serializes the data and Transformer 2 simply transforms them. On the receiver side, the same transformer chain is executed in reverse order with the respective retransformers. From the SWC's point of view it is totally transparent for them which transformer are used or whether transformers are used at all.

A further example of data transformation is shown in Figure 7.2. Here the use-case of intra-ECU data transformation is addressed.

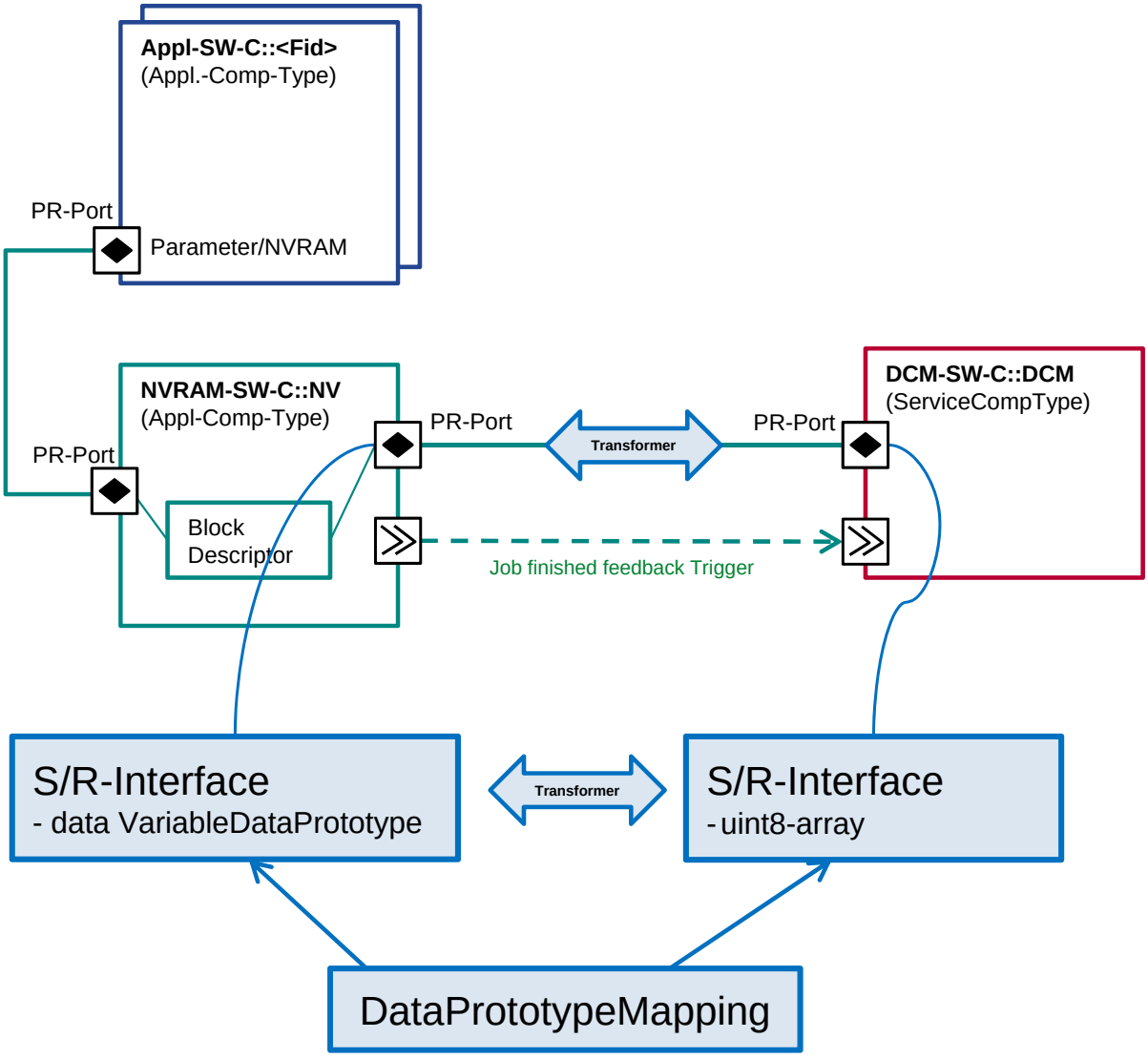

Figure 7.2: Transformer Example for Intra-ECU Communication

The shown intra-ECU transformer is used for converting different representations of data structures between the `NvBlockSwComponentType` and the DCM.

In general transformers have to specify their output format to enable remote ECUs or hardware-dependent BSW modules to correctly work with the transformed data. For that, the serialized (on-wire) format has to be fixed.

Note:
Please be aware that AUTOSAR currently doesn't specify any transformer which only serializes the payload and adds no header in front. The SOME/IP Transformer can serialize all kinds of data but it always adds a partial SOME/IP header in front of the data.

[SWS_Xfrm_00002] ⌈ A transformer shall consider that the target ECU might have a different architecture than the sender ECU (e.g. 8/16/32bit, little/big endian, etc.) so the on-wire format shall be fixed. ⌋*(SRS_Xfrm_00008)*

[SWS_Xfrm_00003] ⌈ A transformer shall clearly define endianness of multi-byte words. ⌋*(SRS_Xfrm_00008)*

[SWS_Xfrm_00004] ⌈ A transformer shall clearly define the ordering of the contained data elements in the complex data if it is a serializer. ⌋*(SRS_Xfrm_00008)*

[SWS_Xfrm_00005] ⌈ A transformer shall clearly define the data semantics. ⌋ *(SRS_Xfrm_00008)* (i.e. representation of data values, e.g. two's complement for signed integers, character encoding for textual data, etc.)

[SWS_Xfrm_00006] ⌈ A transformer shall clearly define the source (=target) data type of the data represented by the byte array if it is a serializer. ⌋*(SRS_Xfrm_00008)*

This is determined by the connected `PortPrototype`/`SystemSignal`.

[SWS_Xfrm_00007] ⌈ A transformer shall clearly define the padding of data. ⌋ *(SRS_Xfrm_00008)*

All of this information is available statically during RTE generation and can therefore be "hardcoded" in the transformer implementation.

A transformer gets its input data via a pointer which destination can vary in length. Therefore, an implementation of a transformer has to cope with input data which are longer than expected.

[SWS_Xfrm_00008] ⌈ The way to deal with unexpected data shall be specified by the transformer specific SWS. In general the transformer shall discard the unexpected data but shall tolerate the expected fraction. ⌋*(SRS_Xfrm_00005)*

This also includes the configurability of the `PortInterfaceMapping` where it can be configured that a sender sends more data than the client receives.

[SWS_Xfrm_00049] ⌈ An implementation of a transformer shall be able to cope with `NULL_PTR` as input data. The detailed behavior shall be specified in the specific transformer SWS. ⌋*(SRS_Xfrm_00005)*

[SWS_Xfrm_00108] ⌈ A transformer which is called with `NULL_PTR` as input data shall not change the output buffer. ⌋*(SRS_Xfrm_00005)*

[SWS_Xfrm_00009] ⌈ A transformer shall be implemented re-entrant because there exist valid configurations which can lead to a concurrent execution of a transformer. ⌋ *(SRS_Xfrm_00006)*

This is independent whether the transformer keeps internal state or not. An explicit synchronization mechanisms inside the transformer might be necessary.

It is possible to configure for a transformer (which is not the first in the the transformer chain of the sending side) to have access to the original data sent by the SWC. This is only supported for the non-first transformers on the sending/calling side (down from

SWC to Rte), not for those on the receiving/called side (up from Rte to SWC). This configuration can be set in the [3, System Template]. The RTE ensures that the original data (which still are placed in the context of the SWC) are not modified by the SWC until the end of the transformer chain.

[SWS_Xfrm_00054] ⌈ If a VariableDataPrototype is mapped to multiple ISignals which referr to DataTransformations and if those DataTransformations referr to the same TransformationTechnologys at the beginning of their list of ordered references transformer and no XfrmVariableDataPrototypeInstanceRef is specified for that TransformationTechnology and no ComBasedTransformer is included in the transformer chains, the execution should be optimzed.

As optimization those first transformers should be executed only once and the result should be taken as input for the further transformers for those ISignals. ⌋ *(SRS_Xfrm_00006)*

[SWS_Xfrm_00101] ⌈ If a Trigger is mapped to multiple ISignals which refer to DataTransformations and if those DataTransformations refer to the same TransformationTechnologys at the beginning of the ordered transformerChain and no XfrmVariableDataPrototypeInstanceRef is specified for that TransformationTechnology and no ComBasedTransformer is included in the transformer chains, the execution should be optimized. ⌋*(SRS_Xfrm_00006)*

If multiple transformer chains in case of a signal fanout in RTE have the same set of transformers at the beginning of the transformer chain, it is possible to optimize and execute those transformers only once for all transformer chains together. The result can be shared between all transformer chains. This is only possible if no ComBasedTransformer is involved.

[SWS_Xfrm_00055] ⌈ If the transformer execution is optimized, the XfrmImplementationMapping shall map all transformers which execution can be optimized to the same BswModuleEntry. ⌋*(SRS_Xfrm_00006)*

If the transformer execution is optimized, the name pattern of the transformer function cannot fulfill the requirements on the name pattern anymore because the same function transforms data for multiple ISignals.

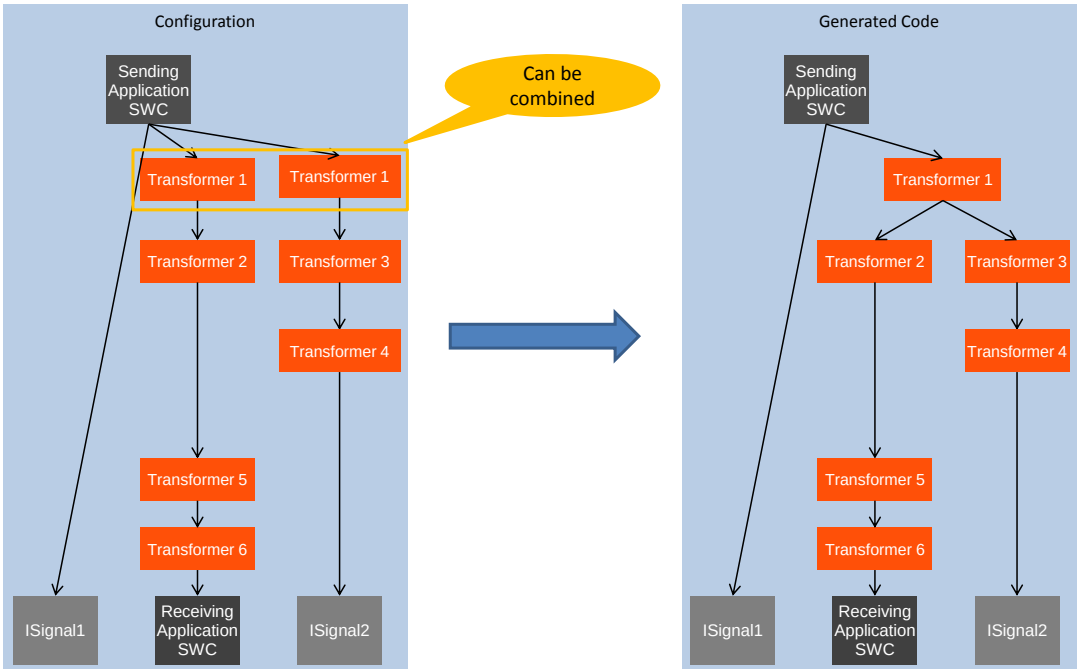

Figure 7.3: Example of a transformer optimization

7.1 Buffer Handling

A transformer will usually work on the data and/or generate some protocol information which are stored in a header and/or footer of the output. Therefore it needs a place to write the result to. Transformers can work with two buffer handling modes: In-place buffer and out-of-place buffer. Which one is used is determined by the configuration in the [3, System Template] and influences the transformer's interface.

[SWS_Xfrm_00010] ⌈ A transformer which uses in-place buffering shall use the input buffer also as output buffer. (See [SWS_Xfrm_00040] and [SWS_Xfrm_00045]) ⌋ *(SRS_Xfrm_00003)*

In this case, the transformation function takes just one buffer pointer argument

[SWS_Xfrm_00011] ⌈ A transformer which uses out-of-place buffering shall work with two buffers: One for the input to the transformer and one for its output. ⌋ *(SRS_Xfrm_00003)*

[SWS_Xfrm_00012] ⌈ A transformer which uses out-of-place buffering shall not alter the data of the input buffer. ⌋ *(SRS_Xfrm_00003)*

The Rte allocates the buffers that are used by the transformers. It calculates the needed buffer size which is needed in worst case for the output. Details for buffer computation are given in [SWS_Rte_03867].

Depending on the specific place of a transformer inside the transformer chain, not all transformers are able to use in-place buffering because a transformer is not allowed modify the original data in the context of the SWC. Also the last transformer on the receiving side cannot use in-place as it has to write its result directly into the buffer of the SWC.

[SWS_Xfrm_00013] ⌈ The first transformer in the chain on the sending side shall use out-of-place buffering. ⌋*(SRS_Xfrm_00003)*

[SWS_Xfrm_00014] ⌈ The last transformer in the chain on the receiving side shall use out-of-place buffering. ⌋*(SRS_Xfrm_00003)*

7.2 Transformer Classes

Different kinds of transformers exist which fulfill totally different functionality. Hence the transformers are categorized into classes.

A transformer class shall contain all transformers which provide similar functionality. At most one transformer of each transformer class shall be allowed per transformer chain.

Currently, the following transformer classes are defined:

- Serializer
- Safety
- Security
- Custom

Further transformer classes might be specified in future AUTOSAR releases.

7.2.1 Serializer

A serializer transformer accepts complex data (either a Sender/Receiver data element or a Client/Server operation with its arguments) or no data (Trigger communication) from the RTE and provides the resulting byte array as an `ISignal` or part of `IPdu`, which is finally transmitted to the receiver by the COM stack.

[SWS_Xfrm_00017] ⌈ A serializer shall take data elements (complex or atomic) and serialize them into a linear representation (byte array). ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_00018] ⌈ The serialization algorithm shall be defined for all possible complex data input. ⌋*(SRS_Xfrm_00001)*

So called "old-world" variable-size array data types are not supported by serializer transformers, only "new-world" variable-size array data types can be transformed. For details, refer to [constr_1387] ([3, System Template]), [TPS_SWCT_01644], [TPS_SWCT_01645], [TPS_SWCT_01642] and [TPS_SWCT_01643].

[SWS_Xfrm_00048] ⌈ A deserializer transformer (serializer transformer on receiver side) shall be able to return all or a subset of the deserialized data to the RTE. ⌋ *(SRS_Xfrm_00001, SRS_Xfrm_00007)*

The [4, SOME/IP Transformer] is a serializer transformer standardized by AUTOSAR.

7.2.2 Safety

A safety transformer protects the communication against unintentional modifications to ensure a safe data transmission.

[SWS_Xfrm_00019] ⌈ A safety transformer shall protect the inter-ECU communication of safety related SWCs. ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_00020] ⌈ A safety transformer shall ensure the correct order of data transmissions. ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_00021] ⌈ A safety transformer shall ensure the correct content of data transmissions. ⌋*(SRS_Xfrm_00001)*

This could be done for example by adding sequence counters and checksums which fulfill the safety requirements.

7.2.3 Security

A security transformer protects the communication against intentional modifications to ensure security of the bus communication.

[SWS_Xfrm_00022] ⌈ A security transformer shall protect the inter-ECU communication of security related SWCs. ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_00023] ⌈ A security transformer shall ensure the authenticity of data transmissions. ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_00024] ⌈ A security transformer shall ensure the integrity of data transmissions. ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_00025] ⌈ A security transformer shall ensure the freshness of data transmissions. ⌋*(SRS_Xfrm_00001)*

This could be done for example by adding sequence counters and checksums which fulfill the security requirements.

7.2.4 Custom

Custom transformers are not specified by AUTOSAR but can be specified by any party in the development workflow to implement a transformer which is not standardized.

Custom transformers can be implemented as CDDs.

7.3 Error Handling

The transformers return errors to the RTE which coordinates the further execution and the notifications of errors up to the SWC.

[SWS_Xfrm_00026] ⌈ Transformers shall return errors to the RTE as return codes. ⌋ *(SRS_Xfrm_00004)*

The RTE decides on the return codes whether to continue the execution of the transformer chain or abort.

There exist two different kinds of transformer errors: Soft Errors and Hard Errors. If a transformer returns a soft error, the Rte continues with the execution of the transformer chain. If a transformer returns a hard error,the Rte aborts the execution of the transformer chain because the error was so severe that there are no meaningful data for the next transformer in the chain.

The value range of errors is divided:

- 0x00: Success
- 0x01 - 0x7F: Soft Errors
- 0x80 - 0xFF: Hard Errors

[SWS_Xfrm_00027] ⌈ If a transformer cannot generate a valid output, it shall return a hard error. ⌋ *(SRS_Xfrm_00004)*

[SWS_Xfrm_00051] ⌈ If a transformer returns a hard error, it shall leave the output buffer unchanged ⌋ *(SRS_Xfrm_00004)*

[SWS_Xfrm_00028] ⌈ If a transformer produces an output but wants to signal warning to the SWC, it shall return a soft error. ⌋ *(SRS_Xfrm_00004)*

For each transformer class, a fixed error set is defined.

[SWS_Xfrm_00029] ⌈ Each transformer class shall have its own set of abstract errors. ⌋ *(SRS_Xfrm_00004, SRS_Xfrm_00010)*

[SWS_Xfrm_00030] ⌈ Each transformer shall return only errors which are a subset of the errors defined for the transformer's transformer class. ⌋ *(SRS_Xfrm_00004, SRS_Xfrm_00010, SRS_Xfrm_00011)*

7.3.1 Errors of Serializer Transformers

[SWS_Xfrm_00031] ⌈ A serializer transformer shall return one of the errors shown in Table 7.1. ⌋*(SRS_Xfrm_00010)*

Error Name	Error Code	Error Type	Description
E_OK	0x00	-	Serialization was successful.
E_NO_DATA	0x01	Soft	No data available which can be deserialized.
Reserved	0x80	Hard	This is reserved to avoid number clashes for autonomous error reactions.
E_SER_GENERIC_ERROR	0x81	Hard	A generic not precisely detailed error occured.
Reserved	0x82 - 0x86	Hard	These are reserved to be compliant with SOME/IP which defines errors with these values that don't relate to serialization and thus can't be created by a transformer.
E_SER_WRONG_ PROTOCOL_VERSION	0x87	Hard	The version of the receiving transformer didn't match the sending transformer.
E_SER_WRONG_ INTERFACE_VERSION	0x88	Hard	Interface version of serialized data is not supported.
E_SER_MALFORMED_ MESSAGE	0x89	Hard	The received message is malformed. The transformer is not able to produce an output.
E_SER_WRONG_ MESSAGE_TYPE	0x8a	Hard	The received message type was not expected.

Table 7.1: Errors of serializer transformers

7.3.2 Errors of Safety Transformers

[SWS_Xfrm_00032] ⌈ A safety transformer shall return one of the errors shown in Table 7.2. ⌋*(SRS_Xfrm_00010)*

Error Name	Error Code	Error Type	Description
E_OK	0x00	-	The communication is safe.
E_SAFETY_VALID_REP	0x01	Soft	The data are valid according to safety, although data with a repeated counter were received.
E_SAFETY_VALID_SEQ	0x02	Soft	The data are valid according to safety, although a counter jump occurred.
E_SAFETY_VALID_ERR	0x03	Soft	The data are valid according to safety, although the check itself failed.
E_SAFETY_VALID_NND	0x05	Soft	Communication is valid according to safety, but no new data received.
E_SAFETY_NODATA_OK	0x20	Soft	No data are available since initialization of transformer.
E_SAFETY_NODATA_REP	0x21	Soft	No data are available since initialization of transformer because a repeated counter was received.

Error Name	Error Code	Error Type	Description
E_SAFETY_NODATA_SEQ	0x22	Soft	No data are available since initialization of transformer and a counter jump occurred.
E_SAFETY_NODATA_ERR	0x23	Soft	No data are available since initialization of transformer. Therefore the check failed.
E_SAFETY_NODATA_NND	0x25	Soft	No data are available since initialization of transformer.
E_SAFETY_INIT_OK	0x30	Soft	Not enough data were received to use them.
E_SAFETY_INIT_REP	0x31	Soft	Not enough data were received to use them but some with a repeated counter were received.
E_SAFETY_INIT_SEQ	0x32	Soft	Not enough data were received to use them, additionally a counter jump occurred.
E_SAFETY_INIT_ERR	0x33	Soft	Not enough data were received to use them, additionally a check failed.
E_SAFETY_INIT_NND	0x35	Soft	Not enough data were received to use them, additionally no new data received.
E_SAFETY_INVALID_OK	0x40	Soft	The data are invalid and cannot be used.
E_SAFETY_INVALID_REP	0x41	Soft	The data are invalid and cannot be used because a repeated counter was received.
E_SAFETY_INVALID_SEQ	0x42	Soft	The data are invalid and cannot be used due to a counter jump.
E_SAFETY_INVALID_ERR	0x43	Soft	The data are invalid and cannot be used because a check failed.
E_SAFETY_INVALID_NND	0x45	Soft	Communication is invalid according to safety and no new data received
E_SAFETY_SOFT_RUNTIMEERROR	0x77	Soft	A runtime error occured, safety properties could not be checked (state or status cannot be determined) but non-protected output data could be produced nonetheless.
E_SAFETY_HARD_RUNTIMEERROR	0xFF	Hard	A runtime error occured, safety properties could not be checked and **no** output data could be produced.

Table 7.2: Errors of safety transformers

Note:
The values 0x04, 0x24, 0x34 and 0x44 are already reserved due to internal use of E2E Library.

7.3.3 Errors of Security Transformers

[SWS_Xfrm_00033] ⌈ A security transformer shall return one of the errors shown in Table 7.3. ⌋*(SRS_Xfrm_00010)*

Error Name	Error Code	Error Type	Description
E_OK	0x00	-	The communication is secure.
E_SEC_NOT_AUTH	0x01	Soft	The data was not authenticated correctly.
E_SEC_NOT_FRESH	0x02	Soft	The data was not fresh.

Table 7.3: Errors of security transformers

7.3.4 Errors of Custom Transformers

[SWS_Xfrm_00050] ⌈ A custom transformer shall return one of the custom errors specified for the custom transformer. See Table 7.4. ⌋*(SRS_Xfrm_00010)*

Error Name	Error Code	Error Type	Description
E_OK	0x00	-	No error occured.
	0x01 - 0x7F	Soft	A transformer specific soft error occured.
	0x80 - 0xFF	Hard	A transformer specific hard error occured.

Table 7.4: Errors of custom transformers

7.4 Development Errors

[SWS_Xfrm_00061] ⌈ A transformer shall support the Development Errors shown in Table 7.5. ⌋*(SRS_BSW_00337)*

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`	`<MIP>_E_UNINIT`	0x01
Error code if an invalid configuration set was selected	`<MIP>_E_INIT_FAILED`	0x02
API service called with wrong parameter	`<MIP>_E_PARAM`	0x03
API service called with invalid pointer	`<MIP>_E_PARAM_POINTER`	0x04

Table 7.5: Development Errors of transformers

where `MIP` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102] totally written in uppercase.

7.5 Production Errors

No production errors are specified for transformers.

7.6 Extended Production Errors

This chapter list and specifies the Extended Production Errors for transformers.

7.6.1 XFRM_E_MALFORMED_MESSAGE

[SWS_Xfrm_00070] ⌈ A transformer shall support the Extended Production Errors shown in Table 7.6. ⌋*(SRS_BSW_00466, SRS_BSW_00469)*

Error Name:	XFRM_E_MALFORMED_MESSAGE	
Short Description:	Transformer not able to produce output due to malformed message content.	
Long Description:	The data handed over to the transformer was malformed. The transformer was not able to produce an output based on the input because it was malformed.	
Detection Criteria:	Fail	The format of the transformer's input doesn't conform to the specification of the specific transformer.
	PASS	The format of the transformer's input conforms to the specification of the specific transformer.
Secondary Parameters:	N/A	
Time Required:	N/A	
Monitor Frequency:	On every execution of transformer.	

Table 7.6: Extended Production Errors of transformers

[SWS_Xfrm_00071] ⌈ The Extended Production Error `XFRM_E_MALFORMED_MESSAGE` shall exist for every transformer which has `XFRM_E_MALFORMED_MESSAGE` set. ⌋*(SRS_BSW_00466, SRS_BSW_00469)*

7.7 Error Notification

Defined in [2, SWS BSW General].

8 API specification

8.1 Imported types

[SWS_Xfrm_00034] ⌈ A transformer shall use the `ImplementationDataType`s defined by RTE in the transformer's Module Interlink Types Header file. ⌋ *(SRS_Xfrm_00002)*

Module Interlink Types Header file, see [SWS_Rte_07503].

8.2 Type definitions

[SWS_Xfrm_00060] ⌈

Name:	`<Mip>_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: <Mip>_ConfigType

⌋ *(SRS_BSW_00404, SRS_BSW_00441)*

8.3 Function definitions

This section defines the generic interfaces of all transformers. These are detailed by the specifications of the specific transformer modules.

[SWS_Xfrm_00062] ⌈ The name pattern `transformerId` should be used for the APIs which belong to the `BswModuleEntry` referenced from a `XfrmImplementationMapping`:

- `Com_<ComSignalName>` if no `XfrmVariableDataPrototypeInstanceRef` exists in the `XfrmImplementationMapping` and `XfrmISignalRef` is used in `XfrmSignal` and the data are sent/received using Com module.
- `Com_<ComSignalGroupName>` if no `XfrmVariableDataPrototypeInstanceRef` exists in the `XfrmImplementationMapping` and `XfrmISignalGroupRef` is used in `XfrmSignal` and the data are sent/received using Com module.
- `LdCom_<LdComIpduName>` if no `XfrmVariableDataPrototypeInstanceRef` exists in the `XfrmImplementationMapping` and the data are sent/received using LdCom module.

- <ComponentName>_<p>_<o> if XfrmVariableDataPrototypeInstanceRef exists.

where

- <ComponentName> is the shortName of the SwComponentPrototype which describes the context of XfrmVariableDataPrototypeInstanceRef.
- <p> is the shortName of the PortPrototype which describes the context of XfrmVariableDataPrototypeInstanceRef. (This is comparable to p used in the RTE APIs.)
- <o> is the shortName of the VariableDataPrototype referenced by XfrmVariableDataPrototypeInstanceRef. (This is comparable to o used in the RTE APIs.)
- <ComSignalName> is the shortName of ComSignal which references the ISignal (using ComSystemTemplateSystemSignalRef that references ISignalToIPduMapping which references the ISignal) that references the DataTransformation.
- <ComSignalGroupName> is the shortName of ComSignalGroup which references the ISignalGroup (using ComSystemTemplateSystemSignalGroupRef that references ISignalToIPduMapping which references the ISignalGroup) that references the DataTransformation.
- <LdComIpduName> is the shortName of LdComIPdu which references the ISignal (using LdComSystemTemplateSignalRef that references ISignalToIPduMapping which references the ISignal) that references the DataTransformation.

⌋(*SRS_Xfrm_00002*)

The name pattern for transformerId is not necessary from the technical point of view to get the transformer working but defines a reliable pattern which simplifies the understandability.

The signature of the transformer function also depends on the configuration parameter XfrmVariableDataPrototypeInstanceRef. If this parameter is used, the SWC, port and data element influence the name of the transformer signature.

This also leads to the generation of multiple transformer functions for one XfrmSignal if the same ISignal or ISignalGroup is referenced by several XfrmImplementationMappings.

8.3.1 <Mip>_<transformerId>

[SWS_Xfrm_00036] ⌈

Service name:	<Mip>_<transformerId>

Syntax:	`uint8 <Mip>_<transformerId>(`
	`uint8* buffer,`
	`uint32* 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 bufferLength	Buffer allocated by the RTE, where the transformed data has to be stored by the transformer Used length of the buffer
Return value:	uint8	0x00 (E_OK): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of the first transformer in a transformer chain of Sender/Receiver communication. 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 8.2: Xfrm_transformerId1

⌋(*SRS_Xfrm_00002*)
where

- `type` is data type of the data element after all data conversion activities of the RTE
- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

This function specified in [SWS_Xfrm_00036] exists on the sender side for each transformed Sender/Receiver communication which uses transformation.

[SWS_Xfrm_00037] ⌈ The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00036] 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`. ⌋(*SRS_Xfrm_00002*)

[SWS_Xfrm_00106] ⌈ The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00036] 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 DataPrototypeMapping in the role firstToSecondDataTransformation. ⌋(*SRS_Xfrm_00002*)

[SWS_Xfrm_00038] ⌈

Service name:	<Mip>_<transformerId>	

Syntax:	`uint8 <Mip>_<transformerId>(`
	`const Rte_Cs_TransactionHandleType* TransactionHandle,`
	`uint8* buffer,`
	`uint32* 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 transformed 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): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of the first transformer in a transformer chain of Client/Server communication. It takes the operation arguments and optionally the return value as input and outputs an uint8 array containing the transformed data. 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 8.3: Xfrm_transformerId2

⌋(*SRS_Xfrm_00002*)

where

- `type` is data type of the data element after all data conversion activities of the RTE
- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

Please note that both the IN and IN/OUT arguments of the `ClientServerOperation` which are transformed are IN arguments from the transformer's point of view because both are only read by the transformer and not written.

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]).

[SWS_Xfrm_00100] ⌈ If the value of the returnValue parameter is inside the range of hard errors (0x80-0xFF), the implementation of [SWS_Xfrm_00038] shall ignore the values of the `ClientServerOperation`'s arguments `data_1`, `...`, `data_n` as they are not filled with meaningful values. ⌋*(SRS_Xfrm_00002)*

[SWS_Xfrm_00039] ⌈ The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00038] 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`. ⌋ *(SRS_Xfrm_00002)*

[SWS_Xfrm_00102] ⌈

Service name:	<Mip>_<transformerId>	

Syntax:	`uint8 <Mip>_<transformerId>(`
	`uint8* buffer,`
	`uint32* bufferLength`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	buffer bufferLength	Buffer allocated by the RTE, where the transformed data has to be stored by the transformer Used length of the buffer
Return value:	uint8	0x00 (E_OK): Transformation successful 0x01 - 0xff: Specific errors

Description:	This function is the interface of the first transformer in a transformer chain of external trigger events. 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: Xfrm_transformerId4

⌋(*SRS_Xfrm_00002*)
where

- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

This function specified in [SWS_Xfrm_00102] exists on the trigger source side for each transformed external trigger event which uses transformation.

[SWS_Xfrm_00103] ⌈ The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00102] 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_Xfrm_00040] ⌈

Service name:	<Mip>_<transformerId>	

Syntax:	`uint8 <Mip>_<transformerId>(`
	`uint8* buffer,`
	`uint32* bufferLength,`
	`[const uint8* inputBuffer,]`
	`uint32 inputBufferLength,`
	`[<type> <originalData>_1,] ...`
	`[<type> <originalData>_n]`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	inputBuffer	This argument only exists for transformers configured for out-of-place transformation. It holds the input data for the transformer.
	inputBufferLength	This argument holds the length of the transformer's input data (in the inputBuffer argument).

	<originalData>_1 ... <originalData>_n	These arguments only exists for transformers on the sending side that are configured for access to the original data. - This denotes the data element represented by the VariableDataPrototype if a Sender/Receiver communication is transformed. - This denotes all arguments of the ClientServerOperation if a Client/Server communication is transformed. ... These arguments only exists for transformers on the sending side that are configured for access to the original data. - This denotes the data element represented by the VariableDataPrototype if a Sender/Receiver communication is transformed. - This denotes all arguments of the ClientServerOperation if a Client/Server communication is transformed.
Parameters (inout):	buffer	This argument is only an INOUT argument for transformers which are not configured for out-of-place transformation. It is the buffer where the input data are placed by the RTE and which is filled by the transformer with its output. This parameter points to the buffer with the output of the previous transformer. If the current transformer has a headerLength different from 0, the output data of the previous transformer begin at position headerLength.
Parameters (out):	buffer bufferLength	This argument is only an OUT argument for transformers configured for out-of-place transformation. It is the buffer allocated by the RTE, where the transformed data has to be stored by the transformer. Used length of the buffer
Return value:	uint8	0x00 (E_OK): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of the first transformer in a transformer chain of Sender/Receiver communication. 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 8.5: Xfrm_transformerId3

⌋(*SRS_Xfrm_00002*)
where

- `type` is data type of the data element after all data conversion activities of the RTE
- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]

- transformerId is the name pattern for the transformer specified in [SWS_Xfrm_00062].

[SWS_Xfrm_00041] ⌈ The function <Mip>_<transformerId> specified in [SWS_Xfrm_00040] shall exist for the non-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. ⌋*(SRS_Xfrm_00002)*

[SWS_Xfrm_00052] ⌈ Each function that satisfies the name pattern <Mip>_<transformerId> (independent from the position in the transformer chain) shall implement its BswModuleEntry which has the same shortName and is referenced by XfrmTransformerBswModuleEntryRef. ⌋*(SRS_Xfrm_00002)*

[SWS_Xfrm_00056] ⌈ If the transformer execution is optimized and one function transforms data (independent from the position in the transformer chain) for multiple ISignals, the <sigName> of the functions name pattern (<Mip>_<transformerId>) may be any shortName of any ISignal which is transformed by that BswModuleEntry. ⌋*(SRS_Xfrm_00006)*

8.3.2 <Mip>_Inv_<transformerId>

[SWS_Xfrm_00042] ⌈

Service name:	<Mip>_Inv_<transformerId>	

Syntax:	`uint8 <Mip>_Inv_<transformerId>(`
	`const uint8* buffer,`
	`uint32 bufferLength,`
	`<type>* dataElement`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	buffer	Buffer allocated by the RTE, where the still serialized data are stored by the Rte. If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, it will hand over a NULL pointer to the transformer.
	bufferLength	Used length of the buffer. If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, the length will be equal to 0.
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): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of a first transformer in a transformer chain of Sender/Receiver communication (this is the last executed transformer on the receiving side!).	

Table 8.6: Xfrm_Inv_transformerId1

⌋(*SRS_Xfrm_00002*)
where

- `type` is data type of the data element before all data conversion activities of the RTE
- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

[SWS_Xfrm_00043] ⌈ The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00042] 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`. ⌋(*SRS_Xfrm_00002*)

[SWS_Xfrm_00107] ⌈ The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00042] 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 `DataPrototypeMapping` in the role `firstToSecondDataTransformation`. ⌋(*SRS_Xfrm_00002*)

[SWS_Xfrm_00044] ⌈

Service name:	<Mip>_Inv_<transformerId>	

Syntax:	`uint8 <Mip>_Inv_<transformerId>(`
	`Rte_Cs_TransactionHandleType* TransactionHandle,`
	`const uint8* buffer,`
	`uint32 bufferLength,`
	`[Std_ReturnType* returnValue,]`
	`[<type>* data_1,] ...`
	`[<type>* data_n]`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	buffer bufferLength	Buffer allocated by the RTE, where the still transformed data are stored by the Rte 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 transformed 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): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of the first transformer in a transformer chain of Client/Server communication (this is the last executed transformer on the receiving side!). It takes the constant buffer (IN parameter buffer) of length (IN parameter bufferLength which may be smaller than the maximum buffer size used by the RTE for buffer allocation) as input and outputs the operation arguments and optionally the return value (OUT parameters data_1, ..., data_n, and returnValue).	

Table 8.7: Xfrm_Inv_transformerId2

⌋(*SRS_Xfrm_00002*)
where

- `type` is data type of the data element before all data conversion activities of the RTE
- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

Please note that both the IN/OUT and OUT arguments of the `ClientServerOperation` which are transformed are OUT arguments from the transformer's point of view because both are only written by the transformer and not read.

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]).

[SWS_Xfrm_00045] ⌈ The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00044] 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`. ⌋
(*SRS_Xfrm_00002*)

[SWS_Xfrm_00104] ⌈

Service name:	<Mip>_Inv_<transformerId>	

Syntax:	`uint8 <Mip>_Inv_<transformerId>(`
	`const uint8* buffer,`
	`uint32 bufferLength`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	buffer bufferLength	Buffer allocated by the RTE, where the still serialized data are stored by the Rte Used length of the buffer
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	0x00 (E_OK): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of a first transformer in a transformer chain of external trigger event communication (this is the last executed transformer on the trigger sink side!).	

Table 8.8: Xfrm_Inv_transformerId4

⌋*(SRS_Xfrm_00002)*
where

- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

This function specified in [SWS_Xfrm_00104] exists on the trigger sink side for each transformed external trigger event which uses transformation.

[SWS_Xfrm_00105] ⌈ The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00104] 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_Xfrm_00046] ⌈

Service name:	<Mip>_Inv_<transformerId>

Syntax:	`uint8 <Mip>_Inv_<transformerId>(`
	`uint8* buffer,`
	`uint32* bufferLength,`
	`[const uint8* inputBuffer,]`
	`uint32 inputBufferLength`
	`)`

Service ID[hex]:	0x04
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	inputBuffer	This argument only exists for transformers configured for out-of-place transformation. It holds the input data for the transformer. If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, it will hand over a NULL pointer to the transformer.
	inputBufferLength	This argument holds the length of the transformer's input data (in the inputBuffer argument). If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, the length will be equal to 0.
Parameters (inout):	buffer	This argument is only an INOUT argument for transformers which are not configured for out-of-place transformation. It is the buffer where the input data are placed by the RTE and which is filled by the transformer with its output. If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, it will hand over a NULL pointer to the transformer.
Parameters (out):	buffer	This argument is only an OUT argument for transformers configured for out-of-place transformation. It is the buffer allocated by the RTE, where the transformed data has to be stored by the transformer.
	bufferLength	Here, the transformer informs the Rte how large the output data really were. It is possible that the length of the output is shorter than the maximum buffer size allocated.
Return value:	uint8	0x00 (E_OK): Transformation successful 0x01 - 0xff: Specific errors
Description:	This function is the interface of a transformer which is not the first transformer in a transformer chain. It takes the output of an earlier transformer in the chain and transforms the data. 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 8.9: Xfrm_Inv_transformerId3

⌋(*SRS_Xfrm_00002*)
where

- `type` is data type of the data element before all data conversion activities of the RTE
- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]
- `transformerId` is the name pattern for the transformer specified in [SWS_Xfrm_00062].

[SWS_Xfrm_00047] ⌈ The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00046] shall exist for the non-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`. ⌋*(SRS_Xfrm_00002)*

[SWS_Xfrm_00053] ⌈ Each function that satisfies the name pattern `<Mip>_Inv_<transformerId>` (independent from the position in the transformer chain) shall implement its `BswModuleEntry` which has the same `shortName` and is referenced by `XfrmInvTransformerBswModuleEntryRef`. ⌋*(SRS_Xfrm_00002)*

8.3.3 <Mip>_Init

[SWS_Xfrm_00058] ⌈

Service name:	<Mip>_Init	

Syntax:	`void <Mip>_Init(`
	`const <Mip>_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 8.10: Xfrm_Init

⌋*(SRS_BSW_00407, SRS_BSW_00411)*
where

- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]

8.3.4 <Mip>_DeInit

[SWS_Xfrm_00059] ⌈

Service name:	<Mip>_DeInit

Syntax:	`void <Mip>_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 8.11: Xfrm_DeInit

⌋(*SRS_BSW_00407*, *SRS_BSW_00411*)
where

- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]

8.3.5 <Mip>_GetVersionInfo

[SWS_Xfrm_00057] ⌈

Service name:	<Mip>_GetVersionInfo	
Syntax:	`void <Mip>_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 8.12: Xfrm_GetVersionInfo

⌋(*SRS_BSW_00407*, *SRS_BSW_00411*)
where

- `Mip` is the Module Implementation Prefix of the transformer as defined in [SWS_BSW_00102]

8.4 Callback notifications

There are no callback notifications.

8.5 Scheduled functions

Transformers have no scheduled functions applicable for all transformers.

8.6 Expected interfaces

There are no expected interfaces.

9 Sequence diagrams

There are no sequence diagrams

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification section 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave section 10.1 in the specification to guarantee comprehension.

Sectin 10.2 specifies the structure (containers) and the parameters of transformers.

Transformer are configured on system level in [3, System Template] and on software component level in [6, Software Component Template]. Out of this information, a basic EcuC of the transformer can be generated.

10.1 How to read this chapter

For details refer to the [2, chapter 10.1 "Introduction to configuration specification" in SWS_BSWGeneral]

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters for a general transformer configuration. The detailed meanings of the parameters describe chapter 7 Functional Specification and chapter 8 API specification.

Specific transformers use this EcuC and fill it with their contents. The EcuC should be created automatically based on the information of `DataTransformationSet` because the generator of a transformer has all necessary information.

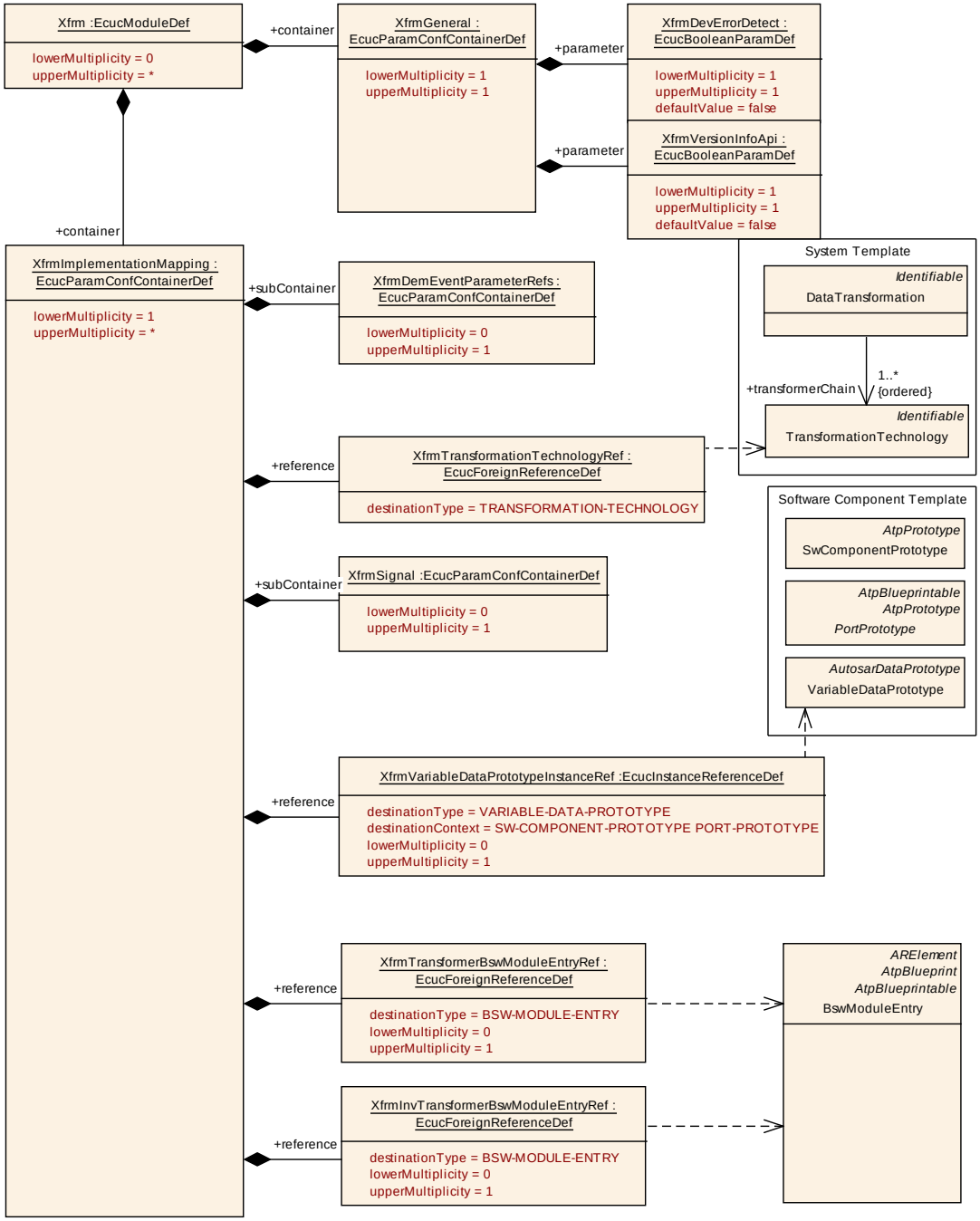

Figure 10.1: AR_EcucDef_Xfrm

Module SWS Item	ECUC_Xfrm_00014	
Module Name	Xfrm	
Module Description		
Post-Build Variant Support	true	
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**

Container Name	Multiplicity	Scope / Dependency
XfrmGeneral	1	Contains the general configuration parameters of the module.
XfrmImplementation Mapping	1..*	For each transformer (TransformationTechnology) in a transformer chain (DataTransformation) which is applied to an ISignal it is necessary to specify the BswModuleEntry which implements it. This is the container to hold these mappings.

10.2.1 XfrmGeneral

SWS Item	[ECUC_Xfrm_00012]
Container Name	XfrmGeneral
Description	Contains the general configuration parameters of the module.
Configuration Parameters	

Name	XfrmDevErrorDetect [ECUC_Xfrm_00013]		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	XfrmVersionInfoApi [ECUC_Xfrm_00019]		
Description	Activate/Deactivate the version information API. • true: version information API activated • false: version information API deactivated		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.2 XfrmImplementationMapping

SWS Item	[ECUC_Xfrm_00001]
Container Name	XfrmImplementationMapping
Description	For each transformer (TransformationTechnology) in a transformer chain (DataTransformation) which is applied to an ISignal it is necessary to specify the BswModuleEntry which implements it. This is the container to hold these mappings.
Configuration Parameters	

Name	XfrmInvTransformerBswModuleEntryRef [ECUC_Xfrm_00005]		
Description	Reference to the BswModuleEntry which implements the referenced inverse transformer on the receiving/called side.		
Multiplicity	0..1		
Type	Foreign reference to BSW-MODULE-ENTRY		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	XfrmTransformationTechnologyRef [ECUC_Xfrm_00003]		
Description	Reference to the TransformationTechnology in the DataTransformation of the system description for which the implementation (BswModuleEntry) shall be mapped.		
Multiplicity	1		
Type	Foreign reference to TRANSFORMATION-TECHNOLOGY		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	XfrmTransformerBswModuleEntryRef [ECUC_Xfrm_00018]		
Description	Reference to the BswModuleEntry which implements the referenced transformer on the sending/calling side.		
Multiplicity	0..1		
Type	Foreign reference to BSW-MODULE-ENTRY		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	XfrmVariableDataPrototypeInstanceRef [ECUC_Xfrm_00011]		
Description	Instance Reference to a VariableDataPrototype in case a dedicated transformer BswModuleEntry is required per VariableDataPrototype access.		
Multiplicity	0..1		
Type	Instance reference to VARIABLE-DATA-PROTOTYPE context: SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
XfrmDemEvent ParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
XfrmSignal	0..1	Reference to the signal in the system description that transports the transformed data.

There are two use cases for the usage of the XfrmVariableDataPrototypeInstanceRef:

1. Transformation of Intra-ECU communication (where no ISignal is available)
2. SWC and port specific transformer functions when one transformer per ISignal is not sufficient. This is the case for E2E protected communication with multiple receivers on the same ECU.

For the transformation of inter-ECU communication, it is necessary to reference the ISignal which transports the data using the XfrmSignal. If intra-ECU communication shall be transformed, no ISignal can be referenced. Therefore it is mandatory to reference the VariableDataPrototype of the affected SWC.

[SWS_Xfrm_CONSTR_09094] ⌈ If there exists a XfrmImplementationMapping which references an ISignal or ISignalGroup *sig1* and contains the optional parameter XfrmVariableDataPrototypeInstanceRef, all XfrmImplementationMappings which reference the same ISignal or ISignalGroup *sig1* shall contain a XfrmVariableDataPrototypeInstanceRef. ⌋*(SRS_Xfrm_00001)*

This means, if XfrmVariableDataPrototypeInstanceRef is used for one transformer in a chain, it also has to be used for all other transformers in that chain.

For E2E protected communication the E2E protection and its verification take place within the E2E transformers. If multiple receivers of the same E2E protected ISignal are located within the same ECU, it is not sufficient to provide one transformer function for verification of the E2E protection on the receiver side. If only one transformer function for the E2E verification would be used for multiple receivers, the same data element would be checked multiple times and the E2E transformer would treat the unchanged sequence number as data duplicates. In this case it is necessary that every local receiver has an own E2E state machine provided to make sure that the accesses to the received data by one receiver don't influence the E2E verification of the data during access by other local receivers of the same data. This can only be realized by providing multiple (port specific) transformer functions for the same ISignal. So every transformer function can maintain its own internal E2E state.

Currently, E2E is the only supported use case for multiple transformer functions of the same ISignal. Due to that multiple transformer functions for port specific transformers are currently only supported for Sender/Receiver communication. The same mechanism can be used in any use case where port specific internal transformer states are needed for Sender/Receiver communication, not only for E2E protected data.

In this case for every VariableDataPrototype referenced by XfrmVariableDataPrototypeInstanceRef a specific transformer function will be generated.

[SWS_Xfrm_CONSTR_09096] ⌈ If no XfrmSignal exists and hence no ISignal or ISignalGroup is referenced, XfrmVariableDataPrototypeInstanceRef shall be used to reference the instance of the VariableDataPrototype which data shall be transformed. ⌋*(SRS_Xfrm_00001)*

[SWS_Xfrm_CONSTR_09095] ⌈ The `XfrmVariableDataPrototypeInstanceRef` shall refer to the instance of a `VariableDataPrototype` which belongs to a subclass of an `AtomicSwComponentType`. ⌋*(SRS_Xfrm_00001)*

This means that `XfrmVariableDataPrototypeInstanceRef` shall referr to a port of a composition.

10.2.3 XfrmSignal

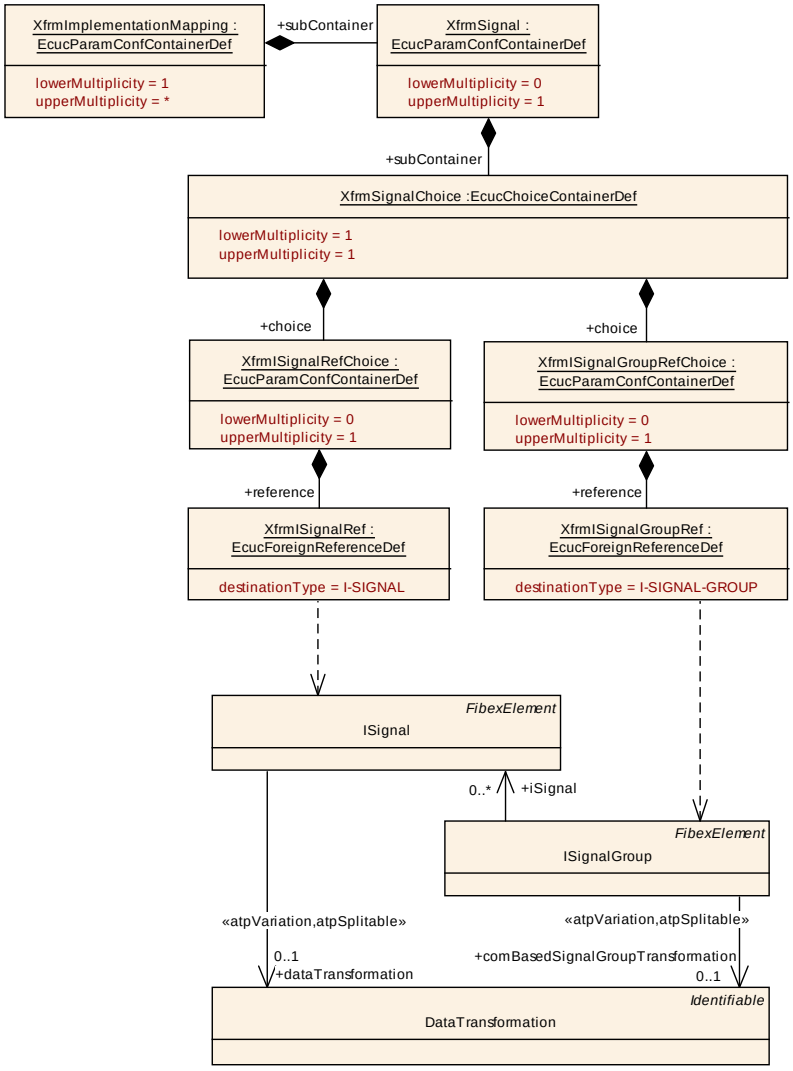

Figure 10.2: AR_EcucDef_XfrmSignal

SWS Item	[ECUC_Xfrm_00002]
Container Name	XfrmSignal
Description	Reference to the signal in the system description that transports the transformed data.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
XfrmSignalChoice	1	Choice whether an ISignal or an ISignalGroup shall be referenced.

SWS Item	[ECUC_Xfrm_00006]
Container Name	XfrmSignalChoice
Description	Choice whether an ISignal or an ISignalGroup shall be referenced.
Configuration Parameters	

Container Choices		
Container Name	**Multiplicity**	**Scope / Dependency**
XfrmISignalGroupRef Choice	0..1	Reference to the ISignalGroup in the system description that transports the transformed data.
XfrmISignalRefChoice	0..1	Reference to the ISignal in the system description that transports the transformed data.

SWS Item	[ECUC_Xfrm_00009]
Container Name	XfrmISignalGroupRefChoice
Description	Reference to the ISignalGroup in the system description that transports the transformed data.
Configuration Parameters	

Name	XfrmISignalGroupRef [ECUC_Xfrm_00010]		
Description	Reference to the ISignalGroup in the system description that transports the transformed data.		
Multiplicity	1		
Type	Foreign reference to I-SIGNAL-GROUP		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

SWS Item	[ECUC_Xfrm_00007]
Container Name	XfrmISignalRefChoice
Description	Reference to the ISignal in the system description that transports the transformed data.
Configuration Parameters	

Name	XfrmISignalRef [ECUC_Xfrm_00008]		
Description	Reference to the ISignal in the system description that transports the transformed data.		
Multiplicity	1		
Type	Foreign reference to I-SIGNAL		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.4 XfrmDemEventParameterRefs

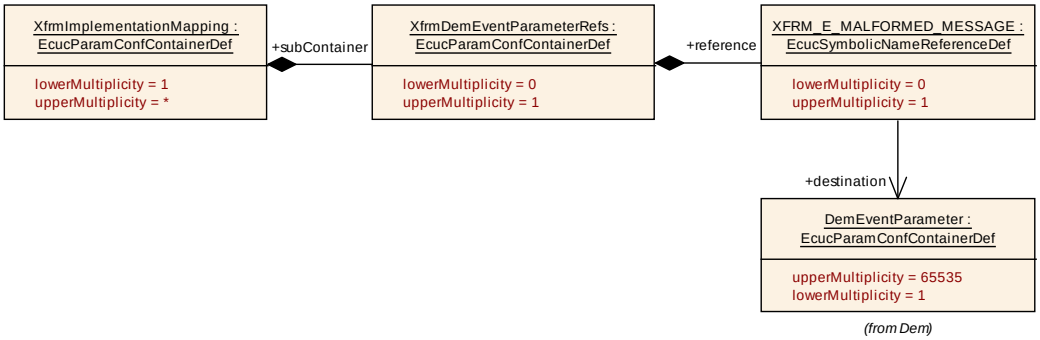

Figure 10.3: AR_EcucDef_XfrmDemEventParameterRefs

SWS Item	[ECUC_Xfrm_00016]
Container Name	XfrmDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

Name	XFRM_E_MALFORMED_MESSAGE [ECUC_Xfrm_00015]
Description	Reference to configured DEM event to report if malformed messages were received by the transformer.
Multiplicity	0..1
Type	Symbolic name reference to DemEventParameter
Post-Build Variant Multiplicity	false

Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: Dem		

No Included Containers

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	**AtomicSwComponentType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, SwComponentType			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
internalBehavior	SwcInternalBehavior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSwComponentType can be located in a different physical file. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSwComponentType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table A.1: AtomicSwComponentType

Class	**BswModuleEntry**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This class represents a single API entry (C-function prototype) into the BSW module or cluster. The name of the C-function is equal to the short name of this element with one exception: In case of multiple instances of a module on the same CPU, special rules for "infixes" apply, see description of class BswImplementation. **Tags:** atp.recommendedPackage=BswModuleEntrys			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=45

Attribute	*Type*	*Mul.*	*Kind*	*Note*
bswEntryKind	BswEntryKindEnum	0..1	attr	This describes whether the entry is concrete or abstract. If the attribute is missing the entry is considered as concrete. **Tags:** xml.sequenceOffset=40
callType	BswCallType	1	attr	The type of call associated with this service. **Tags:** xml.sequenceOffset=25
executionContext	BswExecutionContext	1	attr	Specifies the execution context which is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this service. **Tags:** xml.sequenceOffset=30
functionPrototypeEmitter	NameToken	0..1	attr	This attribute is used to control the generation of function prototypes. If set to "RTE", the RTE generates the function prototypes in the Module Interlink Header File.
isReentrant	Boolean	1	attr	Reentrancy from the viewpoint of function callers: • True: Enables the service to be invoked again, before the service has finished. • False: It is prohibited to invoke the service again before is has finished. **Tags:** xml.sequenceOffset=15
isSynchronous	Boolean	1	attr	Synchronicity from the viewpoint of function callers: • True: This calls a synchronous service, i.e. the service is completed when the call returns. • False: The service (on semantical level) may not be complete when the call returns. **Tags:** xml.sequenceOffset=20
returnType	SwServiceArg	0..1	aggr	The return type belonging to this bswModuleEntry. **Tags:** xml.sequenceOffset=40
role	Identifier	0..1	attr	Specifies the role of the entry in the given context. It shall be equal to the standardized name of the service call, especially in cases where no ServiceIdentifier is specified, e.g. for callbacks. Note that the ShortName is not always sufficient because it maybe vendor specific (e.g. for callbacks which can have more than one instance). **Tags:** xml.sequenceOffset=10

Attribute	*Type*	*Mul.*	*Kind*	*Note*
serviceId	PositiveInteger	0..1	attr	Refers to the service identifier of the Standardized Interfaces of AUTOSAR basic software. For non-standardized interfaces, it can optionally be used for proprietary identification. **Tags:** xml.sequenceOffset=5
swServiceImplPolicy	SwServiceImplPolicyEnum	1	attr	Denotes the implementation policy as a standard function call, inline function or macro. This has to be specified on interface level because it determines the signature of the call. **Tags:** xml.sequenceOffset=35

Table A.2: BswModuleEntry

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	*Type*	*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	*Type*	*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.

Attribute	Type	Mul.	Kind	Note

Table A.4: ClientServerOperation

Class	ClientServerToSignalMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	This element maps the ClientServerOperation to call- and return-SystemSignals.			
Base	ARObject, DataMapping			
Attribute	**Type**	**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.
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	DataPrototypeMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two particular VariableDataPrototypes, ParameterDataPrototypes or ArgumentDataPrototypes with unequal names and/or unequal semantic (resolution or range) in context of two different SenderReceiverInterface, NvDataInterface or ParameterInterface or Operations. If the semantic is unequal following rules apply: The textTableMapping is only applicable if the referred DataPrototypes are typed by AutosarDataType referring to CompuMethods of category TEXTTABLE, SCALE_LINEAR_AND_TEXTTABLE or BITFIELD_TEXTTABLE. In the case that the DataPrototypes are typed by AutosarDataType either referring to CompuMethods of category LINEAR, IDENTICAL or referring to no CompuMethod (which is similar as IDENTICAL) the linear conversion factor is calculated out of the factorSiToUnit and offsetSiToUnit attributes of the referred Units and the CompuRationalCoeffs of a compuInternalToPhys of the referred CompuMethods.			
Base	ARObject			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
firstDataPrototype	AutosarDataPrototype	1	ref	First to be mapped DataPrototype in context of a SenderReceiverInterface, NvDataInterface, ParameterInterface or Operation.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
firstToSecondDataTransformation	DataTransformation	0..1	ref	This defines the need to execute the DataTransformation <Mip>_<transformerId> functions of the transformation chain when communicating from the DataPrototypeMapping.firstDataPrototype to the DataPrototypeMapping.secondDataPrototype. And to execute the DataTransformation <Mip>_Inv_<transformerId> functions of the transformation chain when communicating from the DataPrototypeMapping.secondDataPrototype to the DataPrototypeMapping.firstDataPrototype.
secondDataPrototype	AutosarDataPrototype	1	ref	Second to be mapped DataPrototype in context of a SenderReceiverInterface, NvDataInterface, ParameterInterface or Operation.
subElementMapping	SubElementMapping	*	aggr	This represents the owned SubelementMapping.
textTableMapping	TextTableMapping	0..2	aggr	Applied TextTableMapping(s)

Table A.6: DataPrototypeMapping

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	***Type***	***Mul.***	***Kind***	***Note***
dataTransformationKind	DataTransformationKindEnum	0..1	attr	This attribute controls the kind of DataTransformation to be applied.
executeDespiteDataUnavailability	Boolean	1	attr	Specifies whether the transformer chain is executed even if no input data are available.
transformerChain (ordered)	TransformationTechnology	1..*	ref	This attribute represents the definition of a chain of transformers that are supposed to be executed according to the order of being referenced from DataTransformation.

Table A.7: DataTransformation

Class	**DataTransformationSet**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	This element is the system wide container of DataTransformations which represent transformer chains. **Tags:** atp.recommendedPackage=DataTransformationSets			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataTransformation	DataTransformation	*	aggr	This container consists of all transformer chains which can be used for transformation of data communication. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
transformationTechnology	TransformationTechnology	*	aggr	**Stereotypes:** atpSplitable; atpVariation**Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table A.8: DataTransformationSet

Class	**IPdu (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	The IPdu (Interaction Layer Protocol Data Unit) element is used to sum up all Pdus that are routed by the PduR.			
Base	ARObject, CollectableElement, FibexElement, Identifiable, MultilanguageReferrable, PackageableElement, Pdu, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
containedIPduProps	ContainedIPduProps	0..1	aggr	Defines whether this IPdu may be collected inside a ContainerIPdu.

Table A.9: IPdu

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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataTransformation	DataTransformation	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignal. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataTransformation, variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
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

Attribute	*Type*	*Mul.*	*Kind*	*Note*
iSignalType	ISignalTypeEnum	0..1	attr	This attribute defines whether this iSignal is an array that results in an UINT8_N / UINT8_DYN ComSignalType in the COM configuration or a primitive type.
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.
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.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
timeoutSubstitutionValue	ValueSpecification	0..1	aggr	Defines and enables the ComTimeoutSubstituition for this 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.10: ISignal

Class	**ISignalGroup**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	SignalGroup of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal Group is sent in different SignalIPdus to multiple receivers. An ISignalGroup refers to a set of ISignals that shall always be kept together. A ISignalGroup represents a COM Signal Group. Therefore it is recommended to put the ISignalGroup in the same Package as ISignals (see atp.recommendedPackage) **Tags:** atp.recommendedPackage=ISignalGroup			
Base	ARObject, CollectableElement, FibexElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
comBasedSignalGroupTransformation	DataTransformation	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignalGroup based on the COMBasedTransformer approach. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=comBasedSignalGroupTransformation, variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
iSignal	ISignal	*	ref	Reference to a set of ISignals that shall always be kept together.
systemSignalGroup	SystemSignalGroup	1	ref	Reference to the SystemSignalGroup that is defined on VFB level and that is supposed to be transmitted in the ISignalGroup.
transformationISignalProps	TransformationISignalProps	*	aggr	A transformer chain consists of an ordered list of transformers. The ISignalGroup specific configuration properties for each transformer are defined in the TransformationISignalProps class. The transformer configuration properties that are common for all ISignalGroups are described in the TransformationTechnology class.

Table A.11: ISignalGroup

Class	**ISignalToIPduMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	An ISignalToIPduMapping describes the mapping of ISignals to ISignalIPdus and defines the position of the ISignal within an ISignalIPdu.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
iSignal	ISignal	0..1	ref	Reference to a ISignal that is mapped into the ISignalIPdu. Each ISignal contained in the ISignalGroup shall be mapped into an IPdu by an own ISignalToIPduMapping. The references to the ISignal and to the ISignalGroup in an ISignalToIPduMapping are mutually exclusive.
iSignalGroup	ISignalGroup	0..1	ref	Reference to an ISignalGroup that is mapped into the SignalIPdu. If an ISignalToIPduMapping for an ISignalGroup is defined, only the UpdateIndicationBitPosition and the transferProperty is relevant. The startPosition and the packingByteOrder shall be ignored. Each ISignal contained in the ISignalGroup shall be mapped into an IPdu by an own ISignalToIPduMapping. The references to the ISignal and to the ISignalGroup in an ISignalToIPduMapping are mutually exclusive.
packingByteOrder	ByteOrderEnum	0..1	attr	This parameter defines the order of the bytes of the signal and the packing into the SignalIPdu. The byte ordering "Little Endian" (MostSignificantByteLast), "Big Endian" (MostSignificantByteFirst) and "Opaque" can be selected. For opaque data endianness conversion shall be configured to Opaque. The value of this attribute impacts the absolute position of the signal into the SignalIPdu (see the startPosition attribute description). For an ISignalGroup the packingByteOrder is irrelevant and shall be ignored.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
startPosition	Integer	0..1	attr	This parameter is necessary to describe the bitposition of a signal within an SignalIPdu. It denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" packed signals within the IPdu (see the description of the packingByteOrder attribute). In AUTOSAR the bit counting is always set to "sawtooth" and the bit order is set to "Decreasing". The bit counting in byte 0 starts with bit 0 (least significant bit). The most significant bit in byte 0 is bit 7. Please note that the way the bytes will be actually sent on the bus does not impact this representation: they will always be seen by the software as a byte array. If a mapping for the ISignalGroup is defined, this attribute is irrelevant and shall be ignored.
transferProperty	TransferPropertyEnum	0..1	attr	The triggered or triggeredOnChange, triggeredWithoutRepetition and triggeredOnChangeWithoutRepetition transferProperty causes immediate transmission of the IPdu, except if transmission mode Periodic or transmission mode NONE is defined for the IPdu. The Pending transfer property does not cause transmission of an I-PDU. The immediate transmission of the IPdu is caused even if only one Signal of an IPdu has the transferProperty triggered or triggeredWithoutRepetition or triggeredOnChange or triggeredOnChangeWithoutRepetition and all other Signals have the transferProperty pending. Also for ISignals of an ISignalGroup (GroupSignals) this attribute is relevant and shall be evaluated: • If none of the ISignals belonging to the ISignalGroup have a transferProperty defined the transferProperty of the ISignalToPduMapping referring to the ISignalGroup is considered. • If at least one of the ISignals belonging to the ISignalGroup has a transferProperty defined all other ISignals belonging to the same ISignalGroup shall have a transferProperty defined as well. All of the transferProperties of the GroupSignals are considered.

Attribute	*Type*	*Mul.*	*Kind*	*Note*
updateIndicationBitPosition	Integer	0..1	attr	The UpdateIndicationBit indicates to the receivers that the signal (or the signal group) was updated by the sender. Length is always one bit. The UpdateIndicationBitPosition attribute describes the position of the update bit within the SignalIPdu. For Signals of a ISignalGroup this attribute is irrelevant and shall be ignored. Note that the exact bit position of the updateIndicationBitPosition is linked to the value of the attribute packingByteOrder because the method of finding the bit position is different for the values mostSignificantByteFirst and mostSignificantByteLast. This means that if the value of packingByteOrder is changed while the value of updateIndicationBitPosition remains unchanged the exact bit position of updateIndicationBitPosition within the enclosing ISignalIPdu still undergoes a change. This attribute denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" packed signals within the IPdu (see the description of the packingByteOrder attribute). In AUTOSAR the bit counting is always set to "sawtooth" and the bit order is set to "Decreasing". The bit counting in byte 0 starts with bit 0 (least significant bit). The most significant bit in byte 0 is bit 7.

Table A.12: ISignalToIPduMapping

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, AutosarDataType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.

Attribute	Type	Mul.	Kind	Note
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.13: ImplementationDataType

Class	NvBlockSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The NvBlockSwComponentType defines non volatile data which data can be shared between SwComponentPrototypes. The non volatile data of the NvBlockSwComponentType are accessible via provided and required ports. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement, ARObject, AtomicSwComponentType, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, SwComponentType			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
nvBlockDescriptor	NvBlockDescriptor	*	aggr	Specification of the properties of exactly one NVRAM Block. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.14: NvBlockSwComponentType

Class	PPortPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	ARObject, AbstractProvidedPortPrototype, AtpBlueprintable, AtpFeature, AtpPrototype, Identifiable, MultilanguageReferrable, PortPrototype, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**

Attribute	Type	Mul.	Kind	Note
providedInterface	PortInterface	1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table A.15: PPortPrototype

Class	PortInterfaceMapping (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).			
Base	ARObject, AtpBlueprint, AtpBlueprintable, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table A.16: PortInterfaceMapping

Class	PortPrototype (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	ARObject, AtpBlueprintable, AtpFeature, AtpPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
clientServerAnnotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegatedPortAnnotation	DelegatedPortAnnotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstractionServerAnnotation	IoHwAbstractionServerAnnotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePortAnnotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPortAnnotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPortAnnotation	ParameterPortAnnotation	*	aggr	Annotations on this parameter port.
senderReceiverAnnotation	SenderReceiverAnnotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPortAnnotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

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

Table A.17: PortPrototype

Class	**Referrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	ARObject			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.18: Referrable

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	*Type*	*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.19: 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	*Type*	*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.
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.
systemSignal	SystemSignal	0..1	ref	Reference to the system signal used to carry the primitive ApplicationRecordElement.

Table A.20: 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	*Type*	*Mul.*	*Kind*	*Note*
dataElement	VariableDataPrototype	1..*	aggr	The data elements of this SenderReceiverInterface.
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement

Table A.21: SenderReceiverInterface

Class	**SenderReceiverToSignalMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::DataMapping			
Note	Mapping of a sender receiver communication data element to a signal.			
Base	ARObject, DataMapping			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1	iref	Reference to the data element.
systemSignal	SystemSignal	1	ref	Reference to the system signal used to carry the data element.

Table A.22: SenderReceiverToSignalMapping

Class	**SwComponentPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	ARObject, AtpFeature, AtpPrototype, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
type	SwComponentType	1	tref	Type of the instance. **Stereotypes:** isOfType

Table A.23: SwComponentPrototype

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, MultilanguageReferrable, PackageableElement, Referrable			
Attribute	***Type***	***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.24: SystemSignal

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	***Type***	***Mul.***	***Kind***	***Note***
bufferProperties	BufferProperties	1	aggr	Aggregation of the mandatory BufferProperties.
hasInternalState	Boolean	0..1	attr	This attribute defines whether the Transformer has an internal state or not.
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.25: TransformationTechnology

Class	**Trigger**			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	A trigger which is provided (i.e. released) or required (i.e. used to activate something) in the given context.			
Base	ARObject, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, MultilanguageReferrable, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger.

Table A.26: Trigger

Class	**TriggerInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A trigger interface declares a number of triggers that can be sent by an trigger source. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement, ARObject, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpType, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Port Interface, Referrable			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
trigger	Trigger	1..*	aggr	The Trigger of this trigger interface.

Table A.27: TriggerInterface

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	***Type***	***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.28: TriggerToSignalMapping

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

Table A.29: VariableDataPrototype