Document Title	General Specification of Transformers
Document Category	Software specification (SWS)
Document UID	658
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Changed document type from ASWS to SWS • Removed the <Mip>_GetVersionInfo API • Removed Extended Production Errors • Improved Security Events for Service-Oriented Communication • Removed possibility to pass transformer specific transaction handles
2025-11-27	R25-11	AUTOSAR Release Management	• Added error code E_SER_PAYLOAD_LENGTH_EXCEEDED • Clarified transformer parameters in Client/Server case • Adapted to merge of LdCom into COM • Editorial Changes
2024-11-27	R24-11	AUTOSAR Release Management	• Changed transformers Init and DeInit from "reentrant" to "non reentrant" • ECUC_Xfrm_00014 Supported Config Variants: updated from VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRECOMPILE to VARIANT-PRECOMPILE
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial Changes
2022-11-24	R22-11	AUTOSAR Release Management	• Removed section 8.2.1 Std_TransformerForward • Editorial Changes

▽

△

2021-11-25	R21-11	AUTOSAR Release Management	• Clarification of APIs defined as "Synchronous /Asynchronous" • Contradiction solved in SWS_Xfrm_00108
2020-11-30	R20-11	AUTOSAR Release Management	• Fixed design issues with E2E communication protection for methods • Added Error Codes for E2E • Moved TransformerError and TransformerForward to SWS_StandardTypes • Editorial changes
2019-11-28	R19-11	AUTOSAR Release Management	• Added chapter 8.2.1 Std_TransformerForward • Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Signatures improved • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
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

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 7

2 Acronyms and Abbreviations 8

3 Related documentation 9
- 3.1 Input documents . . . 9
- 3.2 Related standards and norms . . . 10
- 3.3 Related specification . . . 10

4 Constraints and assumptions 11
- 4.1 Limitations . . . 11
- 4.2 Applicability to car domains . . . 11

5 Dependencies to other modules 12
- 5.1 File structure . . . 12
  - 5.1.1 Code file structure . . . 12
  - 5.1.2 Header file structure . . . 12

6 Requirements Tracing 13

7 Functional Specification 15
- 7.1 Buffer Handling . . . 21
- 7.2 Transformer Classes . . . 22
  - 7.2.1 Serializer . . . 23
  - 7.2.2 Safety . . . 23
  - 7.2.3 Security . . . 24
  - 7.2.4 Custom . . . 24
- 7.3 Error Handling . . . 24
  - 7.3.1 Errors of Serializer Transformers . . . 26
  - 7.3.2 Errors of Safety Transformers . . . 27
  - 7.3.3 Errors of Security Transformers . . . 28
  - 7.3.4 Errors of Custom Transformers . . . 29
- 7.4 Error Classification . . . 29
  - 7.4.1 Development Errors . . . 29
  - 7.4.2 Runtime Errors . . . 30
  - 7.4.3 Production Errors . . . 30
- 7.5 Error Notification . . . 30

8 API specification 31
- 8.1 Imported types . . . 31
- 8.2 Type definitions . . . 31
- 8.3 Function definitions . . . 32
  - 8.3.1 <Mip>_ExtractProtocolHeaderFields . . . 33
  - 8.3.2 <Mip>_<transformerId> . . . 34
  - 8.3.3 <Mip>_Inv_<transformerId> . . . 41

8.3.4 <Mip>_Init . . . 47
8.3.5 <Mip>_DeInit . . . 48
8.4 Callback notifications . . . 48
8.5 Scheduled functions . . . 48
8.6 Expected interfaces . . . 48
9 Sequence diagrams 49
10 Configuration specification 50
10.1 How to read this chapter . . . 50
10.2 Containers and configuration parameters . . . 50
10.2.1 XfrmGeneral . . . 52
10.2.2 XfrmImplementationMapping . . . 53
10.2.3 XfrmSignal . . . 58
A Referenced Meta Classes 62
B Change history of AUTOSAR traceable items 81
B.1 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 81
B.1.1 Added Specification Items in R23-11 . . . 81
B.1.2 Changed Specification Items in R23-11 . . . 81
B.1.3 Deleted Specification Items in R23-11 . . . 81
B.2 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 82
B.2.1 Added Specification Items in R24-11 . . . 82
B.2.2 Changed Specification Items in R24-11 . . . 82
B.2.3 Deleted Specification Items in R24-11 . . . 82
B.2.4 Added Constraints in R24-11 . . . 82
B.2.5 Changed Constraints in R24-11 . . . 82
B.2.6 Deleted Constraints in R24-11 . . . 82
B.3 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 83
B.3.1 Added Specification Items in R25-11 . . . 83
B.3.2 Changed Specification Items in R25-11 . . . 83
B.3.3 Deleted Specification Items in R25-11 . . . 83
B.3.4 Added Constraints in R25-11 . . . 83
B.3.5 Changed Constraints in R25-11 . . . 84
B.3.6 Deleted Constraints in R25-11 . . . 84
B.4 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 84
B.4.1 Added Specification Items in R26-11 . . . 84
B.4.2 Changed Specification Items in R26-11 . . . 84
B.4.3 Deleted Specification Items in R26-11 . . . 85
B.4.4 Added Constraints in R26-11 . . . 85
B.4.5 Changed Constraints in R26-11 . . . 85

B.4.6 Deleted Constraints in R26-11 85
C Not applicable requirements 86

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

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] System Template
AUTOSAR_CP_TPS_SystemTemplate

[4] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[5] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[6] Basic Software Module Description Template
AUTOSAR_CP_TPS_BSWModuleDescriptionTemplate

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.

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Xfrm_00058]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Xfrm_00059]
[SRS_BSW_00337]	Classification of development errors	[SWS_Xfrm_00061]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Xfrm_00058] [SWS_Xfrm_00060]
[SRS_BSW_00441]	Naming convention for type, macro and function	[SWS_Xfrm_00060]
[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] [SWS_Xfrm_00112] [SWS_Xfrm_00113] [SWS_Xfrm_00114] [SWS_Xfrm_02000] [SWS_Xfrm_02001] [SWS_Xfrm_02002] [SWS_Xfrm_02003] [SWS_Xfrm_02004] [SWS_Xfrm_02005] [SWS_Xfrm_02006] [SWS_Xfrm_02007] [SWS_Xfrm_02008] [SWS_Xfrm_02009] [SWS_Xfrm_91001] [SWS_Xfrm_91002]
[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] [SWS_Xfrm_CONSTR_09097]
[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]



△

Requirement	Description	Satisfied by
[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]

Table 6.1: Requirements Tracing

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]

Upstream requirements: SRS_Xfrm_00006

⌈Transformers shall be stateful only, if the dedicated transformer functionality requires maintaining a transformer state.⌋

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.

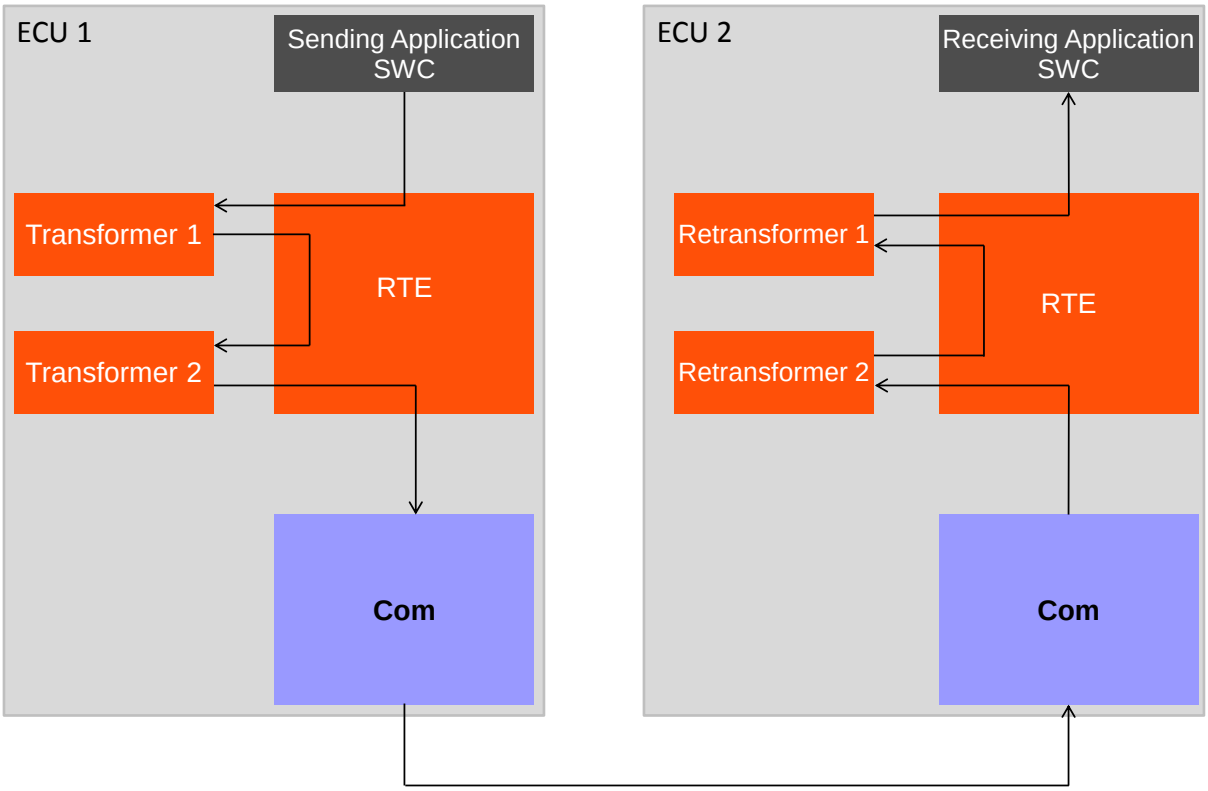

Figure 7.1: Transformer Example for Inter-ECU Communication

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.

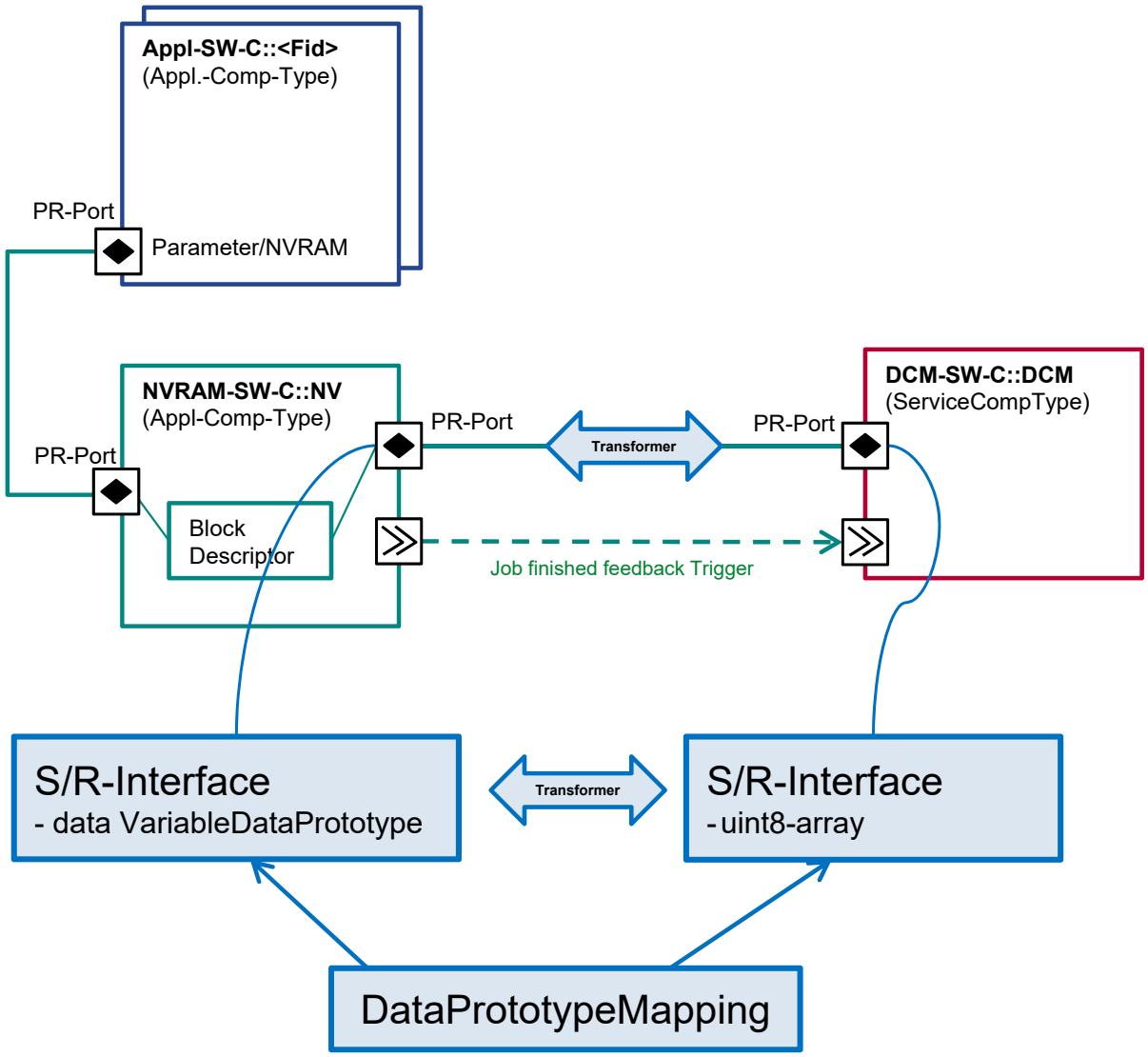

Figure 7.2: Transformer Example for Intra-ECU Communication

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 marshalls the payload and adds no header in front.

[SWS_Xfrm_00002]

Upstream requirements: SRS_Xfrm_00008

⌈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.⌋

[SWS_Xfrm_00003]

Upstream requirements: SRS_Xfrm_00008

⌈A transformer shall clearly define endianness of multi-byte words.⌋

[SWS_Xfrm_00004]

Upstream requirements: SRS_Xfrm_00008

⌈A transformer shall clearly define the ordering of the contained data elements in the complex data if it is a serializer.⌋

[SWS_Xfrm_00005]

Upstream requirements: SRS_Xfrm_00008

⌈A transformer shall clearly define the data semantics.⌋

(i.e. representation of data values, e.g. two's complement for signed integers, character encoding for textual data, etc.)

[SWS_Xfrm_00006]

Upstream requirements: SRS_Xfrm_00008

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

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

[SWS_Xfrm_00007]

Upstream requirements: SRS_Xfrm_00008

⌈A transformer shall clearly define the padding of data.⌋

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]

Upstream requirements: SRS_Xfrm_00005

⌈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.⌋

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]

Upstream requirements: SRS_Xfrm_00005

⌈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.⌋

[SWS_Xfrm_00108]

Upstream requirements: SRS_Xfrm_00005

⌈A transformer which is called with `NULL_PTR` as input data shall not change the output buffer unless the transformer invocation shall trigger an autonomous error reaction (see also [SWS_Rte_07420]).⌋

[SWS_Xfrm_00009]

Upstream requirements: SRS_Xfrm_00006

⌈A transformer shall be implemented re-entrant because there exist valid configurations which can lead to a concurrent execution of a transformer.⌋

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]

Upstream requirements: SRS_Xfrm_00006

⌈If a `VariableDataPrototype` is mapped to multiple `ISignal`s which referr to `DataTransformation`s and if those `DataTransformation`s referr to the same `TransformationTechnology`s at the beginning of their list of ordered references `transformerChain` 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.⌋

[SWS_Xfrm_00101]

Upstream requirements: SRS_Xfrm_00006

⌈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.⌋

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]

Upstream requirements: SRS_Xfrm_00006

⌈If the transformer execution is optimized, the XfrmImplementationMapping shall map all transformers which execution can be optimized to the same BswModuleEntry.⌋

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.

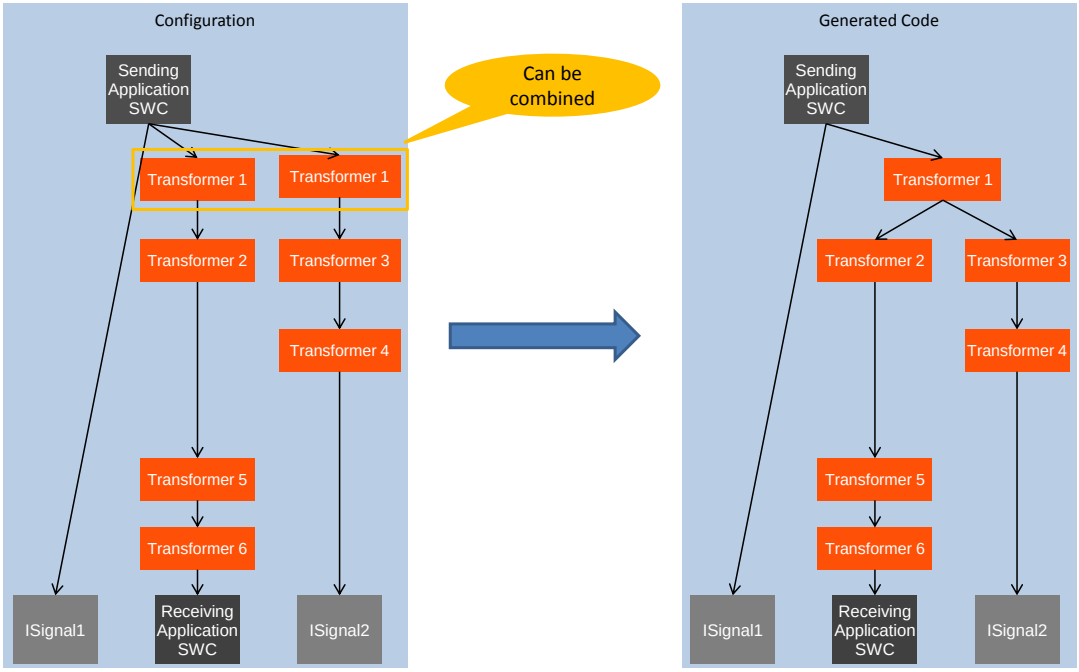

Figure 7.3: Example of a transformer optimization

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]

Upstream requirements: SRS_Xfrm_00003

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

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

[SWS_Xfrm_00011]

Upstream requirements: SRS_Xfrm_00003

⌈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.⌋

[SWS_Xfrm_00012]

Upstream requirements: SRS_Xfrm_00003

⌈A transformer which uses out-of-place buffering shall not alter the data of the input buffer.⌋

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]

Upstream requirements: SRS_Xfrm_00003

⌈The first transformer in the chain on the sending side shall use out-of-place buffering.⌋

[SWS_Xfrm_00014]

Upstream requirements: SRS_Xfrm_00003

⌈The last transformer in the chain on the receiving side shall use out-of-place buffering.⌋

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]

Upstream requirements: SRS_Xfrm_00001

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

[SWS_Xfrm_00018]

Upstream requirements: SRS_Xfrm_00001

⌈The serialization algorithm shall be defined for all possible complex data input.⌋

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] and [TPS_SWCT_01642].

[SWS_Xfrm_00048]

Upstream requirements: SRS_Xfrm_00001, SRS_Xfrm_00007

⌈A deserializer transformer (serializer transformer on receiver side) shall be able to return all or a subset of the deserialized data to the RTE.⌋

7.2.2 Safety

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

[SWS_Xfrm_00019]

Upstream requirements: SRS_Xfrm_00001

⌈A safety transformer shall protect the inter-ECU communication of safety related SWCs.⌋

[SWS_Xfrm_00020]

Upstream requirements: SRS_Xfrm_00001

⌈A safety transformer shall ensure the correct order of data transmissions.⌋

[SWS_Xfrm_00021]

Upstream requirements: SRS_Xfrm_00001

⌈A safety transformer shall ensure the correct content of data transmissions.⌋

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]

Upstream requirements: SRS_Xfrm_00001

⌈A security transformer shall protect the inter-ECU communication of security related SWCs.⌋

[SWS_Xfrm_00023]

Upstream requirements: SRS_Xfrm_00001

⌈A security transformer shall ensure the authenticity of data transmissions.⌋

[SWS_Xfrm_00024]

Upstream requirements: SRS_Xfrm_00001

⌈A security transformer shall ensure the integrity of data transmissions.⌋

[SWS_Xfrm_00025]

Upstream requirements: SRS_Xfrm_00001

⌈A security transformer shall ensure the freshness of data transmissions.⌋

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]

Upstream requirements: SRS_Xfrm_00004

⌈Transformers shall return errors to the RTE as return codes.⌋

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]

Upstream requirements: SRS_Xfrm_00004

⌈If a transformer cannot generate a valid output, it shall return a hard error.⌋

[SWS_Xfrm_00051]

Upstream requirements: SRS_Xfrm_00004

⌈If a transformer returns a hard error, it shall leave the output buffer unchanged⌋

[SWS_Xfrm_00028]

Upstream requirements: SRS_Xfrm_00004

⌈If a transformer produces an output but wants to signal warning to the SWC, it shall return a soft error.⌋

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

[SWS_Xfrm_00029]

Upstream requirements: SRS_Xfrm_00004, SRS_Xfrm_00010

⌈Each transformer class shall have its own set of abstract errors.⌋

[SWS_Xfrm_00030]

Upstream requirements: SRS_Xfrm_00004, SRS_Xfrm_00010, SRS_Xfrm_00011

⌈Each transformer shall return only errors which are a subset of the errors defined for the transformer's transformer class.⌋

Note:
The consequences of the error handling specified here are that soft errors in early stages of a transformer chain (in execution order) might be masked by consecutive hard errors in a later transformer of the chain.

Example:
In case the E2E transformer detects a corrupted (Wrong CRC) or masqueraded (wrong ID/CRC) message, it throws a soft error, while it is possible that the SomeIpXf will override this with a hard error if the message cannot be deserialized. So, a state transition of the E2E state machine might be masked by hard error of deserialization transformer. However, state machine state will stay INVALID as long as messages are invalid, so the INVALID state will be seen by the application once the deserializer is able to deserialize a message.

In such cases, applications that want to rely on the state of E2E transformer state machine only, need to evaluate the hard errors of the deserializer properly in the application.

7.3.1 Errors of Serializer Transformers

[SWS_Xfrm_00031] Errors of serializer transformers

Upstream requirements: 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.
E_SER_PAYLOAD_LENGTH_EXCEEDED	0x40	Soft	The payload length is greater than the expected length of an array.
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.

⌋

7.3.2 Errors of Safety Transformers

[SWS_Xfrm_00032] Errors of safety transformers

Upstream requirements: 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.
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.

Error Name	Error Code	Error Type	Description
E_SAFETY_INVALID_NND	0x45	Soft	Communication is invalid according to safety and no new data received
E_SAFETY_NOSM_OK	0x60	Soft	Communication is safe, Statemachine is not active.
E_SAFETY_NOSM_REP	0x61	Soft	Data with a repeated counter were received. E2EStateMachine disabled.
E_SAFETY_NOSM_SEQ	0x62	Soft	A counter jump occurred. E2EStateMachine disabled.
E_SAFETY_NOSM_ERR	0x63	Soft	The data are invalid and cannot be used because a check failed. E2EStateMachine disabled.
E_SAFETY_NOSM_NND	0x65	Soft	No new data available. E2EStateMachine disabled.
E_SAFETY_NOSM_DEC	0x66	Soft	Disabled E2E State machine and disabled E2E check.
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_E2E_HARD_SAFETY_ERR	0x8d	Hard	Not further specified E2E error
E_SAFETY_HARD_RUNTIMEERROR	0xFF	Hard	A runtime error occured, safety properties could not be checked and **no** output data could be produced.

⌋

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] Errors of security transformers

Upstream requirements: 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.

⌋

7.3.4 Errors of Custom Transformers

[SWS_Xfrm_00050] Errors of custom transformers

Upstream requirements: 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.

⌋

7.4 Error Classification

Chapter [2, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of the following error types which may occur in BSW modules.

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

7.4.1 Development Errors

[SWS_Xfrm_00061] Definition of development errors in module Xfrm

Upstream requirements: SRS_BSW_00337

⌈

Type of error	*Related error code*	*Error value*
Error code if any API service is called before the transformer module was initialized with Init or after a call to DeInit	<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

⌋

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

7.4.2 Runtime Errors

There are no runtime errors.

7.4.3 Production Errors

There are no production errors.

7.5 Error Notification

Defined in [2, SWS BSW General].

8 API specification

8.1 Imported types

[SWS_Xfrm_00034]

Upstream requirements: SRS_Xfrm_00002

⌈A transformer shall use the `ImplementationDataType`s defined by RTE in the transformer's Module Interlink Types Header file.⌋

Module Interlink Types Header file, see [SWS_Rte_07503].

A transformer shall further use the types defined in the following table.

[SWS_Xfrm_91001] Definition of imported datatypes of module Xfrm

Upstream requirements: SRS_Xfrm_00002

⌈

Module	*Header File*	*Imported Type*
Rte	SchM_<Mip>.h	Rte_Cs_TransactionHandleType
Std	Std_Types.h	Std_ExtractProtocolHeaderFieldsType
	Std_Types.h	Std_MessageResultType
	Std_Types.h	Std_MessageTypeType
	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_TransformerForwardCode (draft)

⌋

8.2 Type definitions

[SWS_Xfrm_00060] Definition of datatype {Mip}_ConfigType

Upstream requirements: SRS_BSW_00404, SRS_BSW_00441

⌈

Name	{Mip}_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	–
Description	This is the type of the data structure containing the initialization data for the transformer.	
Available via	<Mip>.h	

⌋

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]

Upstream requirements: SRS_Xfrm_00002

⌈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 is sent/received using Com module and `ComIPduLargeData` of the `ComIPdu` containing the `ComSignal` of the referenced `ISignal` is not set to `TRUE`.
- `Com_<ComSignalGroupName>` if no `XfrmVariableDataPrototypeInstanceRef` exists in the `XfrmImplementationMapping` and `XfrmISignalGroupRef` is used in `XfrmSignal` and the data is sent/received using Com module and `ComIPduLargeData` of the `ComIPdu` containing the `ComSignalGroup` of the referenced `ISignalGroup` is not set to `TRUE`.
- `Com_<ComIPduName>` if no `XfrmVariableDataPrototypeInstanceRef` exists in the `XfrmImplementationMapping` and the data is sent/received using Com module and `ComIPduLargeData` of the `ComIPdu` is set to `TRUE`.
- `<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 the `ComSignal` which references the `ISignal` (using `ComSignal.ComSystemTemplateSystemSignalRef` that references the `ISignalToIPduMapping` which references the `ISignal`) that references the `DataTransformation`.

- <ComSignalGroupName> is the shortName of the ComSignalGroup which references the ISignalGroup (using ComSignalGroup.ComSystemTemplateSignalGroupRef that references the ISignalToIPduMapping which references the ISignalGroup) that references the DataTransformation.
- <ComIPduName> is the shortName of the ComIPdu which references the ISignal (via ComIPdu.ComIPduSignalRef.ComSystemTemplateSystemSignalRef that references the ISignalToIPduMapping which references the ISignal) that references the DataTransformation.

⌋

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>_ExtractProtocolHeaderFields

[SWS_Xfrm_91002] Definition of API function <Mip>_ExtractProtocolHeaderFields

Upstream requirements: SRS_Xfrm_00002

⌈

Service Name	<Mip>_ExtractProtocolHeaderFields	

Syntax	`Std_ReturnType <Mip>_ExtractProtocolHeaderFields (`
	`  const uint8* buffer,`
	`  uint32 bufferLength,`
	`  Std_MessageTypeType* messageType,`
	`  Std_MessageResultType* messageResult`
	`)`	

Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	buffer	Buffer allocated by the RTE, where the transformed data has to be stored by the transformer
	bufferLength	Length of the buffer
Parameters (inout)	None	

▽

△

Parameters (out)	messageType	Canonical representation of the message type (extracted from the transformers protocol header).
	messageResult	Canonical representation of the message result type (extracted from the transformers protocol header).
Return value	Std_ReturnType	E_OK: Relevant protocol header fields have been extracted successfully. E_NOT_OK: An error occurred during parsing of the protocol header.
Description	Function to extract the relevant protocol header fields of the message and the type of the message result of a transformer. - At the time being, this is limited to the types used for C/S communication (i.e., REQUEST and RESPONSE and OK and ERROR).	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_00112]

Upstream requirements: SRS_Xfrm_00002

⌈The function <Mip>_ExtractProtocolHeaderFields specified in [SWS_Xfrm_91002] shall exist in case the respective transformer processes relevant protocol header fields related to the type of a message and the type of the message result. – This function shall extract this information and provide it in a canonical representation via its output arguments.⌋

[SWS_Xfrm_00113]

Upstream requirements: SRS_Xfrm_00002

⌈The function <Mip>_ExtractProtocolHeaderFields specified in [SWS_Xfrm_91002] shall return E_NOT_OK in case of an error (e.g., parsing error) during extraction. Neither messageType nor messageResult shall be modified in this case.⌋

[SWS_Xfrm_00114]

Upstream requirements: SRS_Xfrm_00002

⌈The function <Mip>_ExtractProtocolHeaderFields specified in [SWS_Xfrm_91002] shall return E_OK otherwise.⌋

8.3.2 <Mip>_<transformerId>

[SWS_Xfrm_00036] Definition of API function <Mip>_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

Service Name	<Mip>_<transformerId>	
Syntax	`uint8 <Mip>_<transformerId> ( uint8* buffer, uint32* bufferLength, <paramtype> dataElement )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dataElement	Data element which shall be transformed
Parameters (inout)	None	
Parameters (out)	buffer	Buffer allocated by the RTE, where the transformed data has to be stored by the transformer
	bufferLength	Used length of the buffer
Return value	uint8	0x00 (`E_OK`): 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.	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02000] Variable prototype parts of [SWS_Xfrm_00036]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00036] shall use the following rules to derive its variable prototype parts:

- `paramtype` is derived from `type` according to the parameter passing rules rules defined by the [4, SRS BSW General] (see [SRS_BSW_00484], [SRS_BSW_00485], and [SRS_BSW_00486]) and [2, SWS BSW General] (see [SWS_BSW_00186]).
- `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]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00036] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `SenderReceiverToSignalMapping`, a `SenderRecRecordElementMapping` or a `SenderRecArrayElementMapping`.⌋

[SWS_Xfrm_00106]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00036] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `DataPrototypeMapping` in the role `firstToSecondDataTransformation`.⌋

[SWS_Xfrm_00038] Definition of API function <Mip>_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

Service Name	<Mip>_<transformerId>	

Syntax	`uint8 <Mip>_<transformerId> (`
	`  const Rte_Cs_TransactionHandleType* TransactionHandle,`
	`  uint8* buffer,`
	`  uint32* bufferLength,`
	`  [Std_ReturnType returnValue],`
	`  [<paramtype> data_1, ...`
	`  <paramtype> 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 a 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.	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02001] Variable prototype parts of [SWS_Xfrm_00038]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00038] shall use the following rules to derive its variable prototype parts:

- `paramtype` is derived from `type` according to the parameter passing rules rules defined by the [4, SRS BSW General] (see [SRS_BSW_00484], [SRS_BSW_00485], and [SRS_BSW_00486]) and [2, SWS BSW General] (see [SWS_BSW_00186]).
- `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.

[SWS_Xfrm_00100]

Upstream requirements: SRS_Xfrm_00002

⌈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.⌋

[SWS_Xfrm_00039]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00038] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `ClientServerToSignalMapping` in the `callSignal` or `returnSignal`.⌋

[SWS_Xfrm_00102] Definition of API function <Mip>_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

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	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 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.	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02002] Variable prototype parts of [SWS_Xfrm_00102]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00102] shall use the following rules to derive its variable prototype parts:

- `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]

Upstream requirements: SRS_Xfrm_00002

⌈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`.⌋

[SWS_Xfrm_00040] Definition of API function <Mip>_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

Service Name	<Mip>_<transformerId>	

Syntax	`uint8 <Mip>_<transformerId> (`
	`  [Std_TransformerForwardCode forwardedCode],`
	`  [Std_ExtractProtocolHeaderFieldsType extractProtocolHeaderFields],`
	`  uint8* buffer,`
	`  uint32* bufferLength,`
	`  [const uint8* inputBuffer],`
	`  uint32 inputBufferLength,`
	`  [<paramtype> originalData]`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant Depends on specific transformer	
Parameters (in)	forwardedCode	Optional forwarded transformer code. This argument only exists if the corresponding PortPrototype is referenced by PortAPIOption with transformerStatusForwarding set to transformerStatus Forwarding.
	extractProtocolHeader Fields	Optional pointer to the function that shall be used to extract relevant protocol header fields of a previous transformer in the transformer chain. This argument only exists if the corresponding XfrmImplementationMapping has a XfrmTransformerClassExtract ProtocolHeaderFields.
	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	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 VariableData Prototype 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	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	Used length of the buffer
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 of Sender/Receiver or Client/Server 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.	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02003] Variable prototype parts of [SWS_Xfrm_00040]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00040] shall use the following rules to derive its variable prototype parts:

- `paramtype` is derived from `type` according to the parameter passing rules rules defined by the [4, SRS BSW General] (see [SRS_BSW_00484], [SRS_BSW_00485], and [SRS_BSW_00486]) and [2, SWS BSW General] (see [SWS_BSW_00186]).
- `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]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_<transformerId>` specified in [SWS_Xfrm_00040] shall exist for the non-first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation`.⌋

[SWS_Xfrm_00052]

Upstream requirements: SRS_Xfrm_00002

⌈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.⌋

That means that XfrmTransformerBswModuleEntryRef has to exist in any case if this transformer is used on sender side. It can only be omitted if the transformer is only used on receiver side.

[SWS_Xfrm_00056]

Upstream requirements: SRS_Xfrm_00006

⌈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.⌋

8.3.3 <Mip>_Inv_<transformerId>

[SWS_Xfrm_00042] Definition of API function <Mip>_Inv_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

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)	dataElement	Data element which is the result of the transformation and contains the deserialized data element
Parameters (out)	None	
Return value	uint8	0x00 (E_OK): 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!).
Available via	<Mip>.h

⌋

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

[SWS_Xfrm_02004] Variable prototype parts of [SWS_Xfrm_00042]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00042] shall use the following rules to derive its variable prototype parts:

- `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]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00042] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `SenderReceiverToSignalMapping`, a `SenderRecRecordElementMapping` or a `SenderRecArrayElementMapping`.⌋

[SWS_Xfrm_00107]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00042] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `DataPrototypeMapping` in the role `firstToSecondDataTransformation`.⌋

[SWS_Xfrm_00044] Definition of API function <Mip>_Inv_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

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	Buffer allocated by the RTE, where the still transformed data are stored by the Rte
	bufferLength	Used length of the buffer
Parameters (inout)	data_1	Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface)
	...	...
	data_n	Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface)
Parameters (out)	TransactionHandle	Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests.
	returnValue	Return value 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 ClientServer Operation has at least one PossibleError defined.
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).	
Available via	<Mip>.h	

⌋

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

[SWS_Xfrm_02005] Variable prototype parts of [SWS_Xfrm_00044]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00044] shall use the following rules to derive its variable prototype parts:

- `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.

[SWS_Xfrm_00045]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00044] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `ClientServerToSignalMapping` in the `callSignal` or `returnSignal`.⌋

[SWS_Xfrm_00104] Definition of API function <Mip>_Inv_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

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	Buffer allocated by the RTE, where the still serialized data are stored by the Rte
	bufferLength	Used length of the buffer
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	0x00 (`E_OK`): 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!).	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02006] Variable prototype parts of [SWS_Xfrm_00104]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00104] shall use the following rules to derive its variable prototype parts:

- `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]

Upstream requirements: SRS_Xfrm_00002

⌈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`.⌋

[SWS_Xfrm_00046] Definition of API function <Mip>_Inv_<transformerId>

Upstream requirements: SRS_Xfrm_00002

⌈

Service Name	<Mip>_Inv_<transformerId>	

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

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant Depends on specific transformer	
Parameters (in)	extractProtocolHeader Fields	Optional pointer to the function that shall be used to extract relevant protocol header fields of a previous transformer in the transformer chain. This argument only exists if the corresponding XfrmImplementationMapping has a XfrmTransformerClassExtract ProtocolHeaderFields.

▽

△

	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 executeDespiteData Unavailability 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.	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02007] Variable prototype parts of [SWS_Xfrm_00046]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00046] shall use the following rules to derive its variable prototype parts:

- `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]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_Inv_<transformerId>` specified in [SWS_Xfrm_00046] shall exist for the non-first reference in the list of ordered references `transformerChain`

from a DataTransformation to a TransformationTechnology if the DataTransformation is referenced by an ISignal in the role dataTransformation.⌋

[SWS_Xfrm_00053]

Upstream requirements: SRS_Xfrm_00002

⌈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.⌋

That means that XfrmInvTransformerBswModuleEntryRef has to exist in any case if this transformer is used on receiver side. It can only be omitted if the transformer is only used on sender side.

8.3.4 <Mip>_Init

[SWS_Xfrm_00058] Definition of API function <Mip>_Init

Upstream requirements: SRS_BSW_00101, SRS_BSW_00404

⌈

Service Name	<Mip>_Init	

Syntax	`void <Mip>_Init (`
	`  const {Mip}_ConfigType* config`
	`)`	

Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	config	Pointer to the transformer's configuration data.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service initializes the transformer for the further processing.	
Available via	<Mip>.h	

⌋

[SWS_Xfrm_02008] Variable prototype parts of [SWS_Xfrm_00058]

Upstream requirements: SRS_Xfrm_00002

⌈The function <Mip>_Init specified in [SWS_Xfrm_00058] shall use the following rules to derive its variable prototype parts:

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

⌋

8.3.5 <Mip>_DeInit

[SWS_Xfrm_00059] Definition of API function <Mip>_DeInit

Upstream requirements: SRS_BSW_00336

⌈

Service Name	<Mip>_DeInit

Syntax	`void <Mip>_DeInit (`
	`  void`
	`)`

Service ID [hex]	0x02
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This service deinitializes the transformer.
Available via	<Mip>.h

⌋

[SWS_Xfrm_02009] Variable prototype parts of [SWS_Xfrm_00059]

Upstream requirements: SRS_Xfrm_00002

⌈The function `<Mip>_DeInit` specified in [SWS_Xfrm_00059] shall use the following rules to derive its variable prototype parts:

- `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 Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 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 [5, 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"*

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.

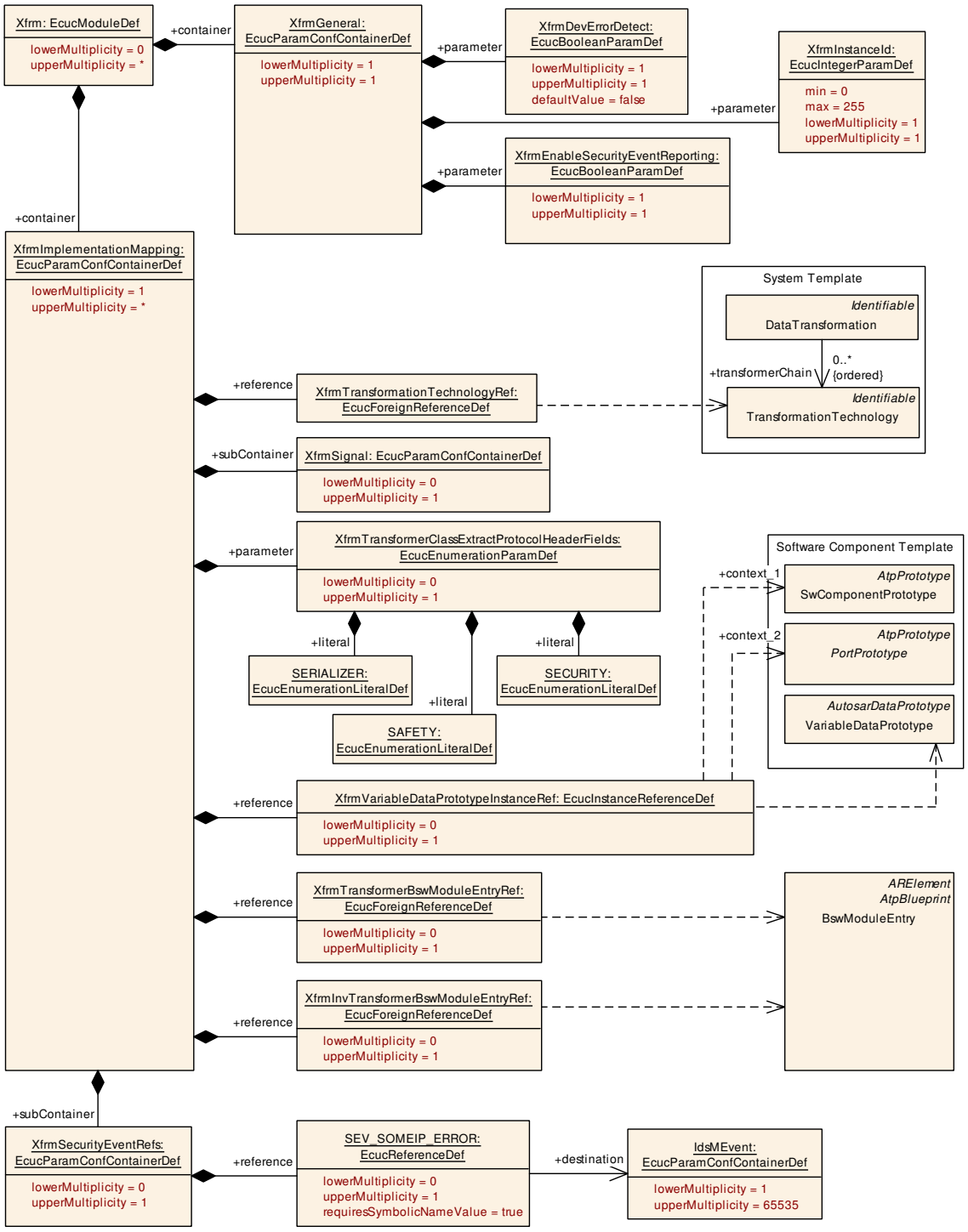

Figure 10.1: AR_EcucDef_Xfrm

[ECUC_Xfrm_00014] Definition of EcucModuleDef Xfrm ⌈

Module Name	Xfrm
Description	Configuration of the Xfrm module.
Post-Build Selectable Support	false
Post-Build Loadable Support	false

Included Containers		
Container Name	**Multiplicity**	**Description**
XfrmGeneral	1	Contains the general configuration parameters of the module.
XfrmImplementationMapping	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

[ECUC_Xfrm_00012] Definition of EcucParamConfContainerDef XfrmGeneral ⌈

Container Name	XfrmGeneral
Parent Container	Xfrm
Upstream mapping defined	–
Description	Contains the general configuration parameters of the module.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
XfrmDevErrorDetect	1	[ECUC_Xfrm_00013]
XfrmEnableSecurityEventReporting	1	[ECUC_Xfrm_00023]
XfrmInstanceId	1	[ECUC_Xfrm_00020]

No Included Containers

⌋

[ECUC_Xfrm_00013] Definition of EcucBooleanParamDef XfrmDevErrorDetect ⌈

Parameter Name	XfrmDevErrorDetect
Parent Container	XfrmGeneral
Upstream mapping defined	–
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
Pre-Compile only	true

⌋

[ECUC_Xfrm_00023] Definition of EcucBooleanParamDef XfrmEnableSecurityEventReporting ⌈

Parameter Name	XfrmEnableSecurityEventReporting
Parent Container	XfrmGeneral
Upstream mapping defined	–
Description	Switches the reporting of security events to the IdsM: • true: reporting is enabled. • false: reporting is disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Xfrm_00020] Definition of EcucIntegerParamDef XfrmInstanceId ⌈

Parameter Name	XfrmInstanceId	
Parent Container	XfrmGeneral	
Upstream mapping defined	–	
Description	Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Pre-Compile only	true	

⌋

10.2.2 XfrmImplementationMapping

[ECUC_Xfrm_00001] Definition of EcucParamConfContainerDef XfrmImplementationMapping ⌈

Container Name	XfrmImplementationMapping
Parent Container	Xfrm
Upstream mapping defined	–
Description	For each transformer (TransformationTechnology) in a transformer chain (Data Transformation) which is applied to an ISignal it is necessary to specify the BswModule Entry which implements it. This is the container to hold these mappings.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
XfrmTransformerClassExtractProtocolHeaderFields	0..1	[ECUC_Xfrm_00022]
XfrmInvTransformerBswModuleEntryRef	0..1	[ECUC_Xfrm_00005]
XfrmTransformationTechnologyRef	1	[ECUC_Xfrm_00003]
XfrmTransformerBswModuleEntryRef	0..1	[ECUC_Xfrm_00018]
XfrmVariableDataPrototypeInstanceRef	0..1	[ECUC_Xfrm_00011]

Included Containers		
Container Name	**Multiplicity**	**Description**
XfrmSecurityEventRefs	0..1	Container for the references to IdsMEvent elements representing the security events that the Transformer module shall report to the IdsM in case the corresponding security related event occurs (and if XfrmEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events.
XfrmSignal	0..1	Reference to the signal in the system description that transports the transformed data.

⌋

[ECUC_Xfrm_00022] Definition of EcucEnumerationParamDef XfrmTransformerClassExtractProtocolHeaderFields ⌈

Parameter Name	XfrmTransformerClassExtractProtocolHeaderFields	
Parent Container	XfrmImplementationMapping	
Upstream mapping defined	–	
Description	Defines the transformerClass of the TransformationTechnology containing information in its protocol header that is required to distinguish between requests vs. responses and normal vs. error responses in C/S communication. Usually this shall be the TransformationTechnology with transformerClass equal to "serializer". Setting this parameter basically instructs the RTE to pass a pointer to the Mip_ExtractProtocol HeaderFields() function of the respective transformer as an additional argument to the called transformer function. E.g., if the serializing transformer in the transformer chain is SomeIpXf and this parameter is set to SERIALIZER, then SomeIpXf_ExtractProtocol HeaderFields() will be passed as additional argument.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	SAFETY	The Mip_ExtractProtocolHeaderFields function of the safety transformer in the chain shall be called.
	SECURITY	The Mip_ExtractProtocolHeaderFields function of the security transformer in the chain shall be called.
	SERIALIZER	The Mip_ExtractProtocolHeaderFields function of the serializing transformer in the chain shall be called
Pre-Compile only	true	

⌋

[ECUC_Xfrm_00005] Definition of EcucForeignReferenceDef XfrmInvTransformerBswModuleEntryRef ⌈

Parameter Name	XfrmInvTransformerBswModuleEntryRef
Parent Container	XfrmImplementationMapping
Upstream mapping defined	[6, CP_TPS_BSWModuleDescriptionTemplate]
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
Pre-Compile only	true

⌋

[ECUC_Xfrm_00003] Definition of EcucForeignReferenceDef XfrmTransformationTechnologyRef ⌈

Parameter Name	XfrmTransformationTechnologyRef
Parent Container	XfrmImplementationMapping
Upstream mapping defined	[3, CP_TPS_SystemTemplate]
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
Pre-Compile only	true

⌋

[ECUC_Xfrm_00018] Definition of EcucForeignReferenceDef XfrmTransformer BswModuleEntryRef ⌈

Parameter Name	XfrmTransformerBswModuleEntryRef
Parent Container	XfrmImplementationMapping
Upstream mapping defined	[6, CP_TPS_BSWModuleDescriptionTemplate]
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
Pre-Compile only	true

⌋

[ECUC_Xfrm_00011] Definition of EcucInstanceReferenceDef XfrmVariableData PrototypeInstanceRef ⌈

Parameter Name	XfrmVariableDataPrototypeInstanceRef
Parent Container	XfrmImplementationMapping
Upstream mapping defined	[3, CP_TPS_SystemTemplate]
Description	Instance Reference to a VariableDataPrototype in case a dedicated transformer Bsw ModuleEntry is required per VariableDataPrototype access.

▽

△

Multiplicity	0..1
Type	Instance reference to VARIABLE-DATA-PROTOTYPE context: SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE
Pre-Compile only	true

⌋

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]

Upstream requirements: SRS_Xfrm_00001

⌈If there exists a `XfrmImplementationMapping` which references an `ISignal` or `ISignalGroup` *sig1* and contains the optional parameter `XfrmVariableDataPrototypeInstanceRef`, all `XfrmImplementationMapping`s which reference the same `ISignal` or `ISignalGroup` *sig1* shall contain a `XfrmVariableDataPrototypeInstanceRef`.⌋

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]

Upstream requirements: SRS_Xfrm_00001

⌈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.⌋

[SWS_Xfrm_CONSTR_09095]

Upstream requirements: SRS_Xfrm_00001

⌈The XfrmVariableDataPrototypeInstanceRef shall refer to the instance of a VariableDataPrototype which belongs to a subclass of an AtomicSwComponentType.⌋

[ECUC_Xfrm_00024] Definition of EcucParamConfContainerDef XfrmSecurityEventRefs ⌈

Container Name	XfrmSecurityEventRefs
Parent Container	XfrmImplementationMapping
Upstream mapping defined	–
Description	Container for the references to IdsMEvent elements representing the security events that the Transformer module shall report to the IdsM in case the corresponding security related event occurs (and if XfrmEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SEV_SOMEIP_ERROR	0..1	[ECUC_Xfrm_00025]

No Included Containers

⌋

[ECUC_Xfrm_00025] Definition of EcucReferenceDef SEV_SOMEIP_ERROR ⌈

Parameter Name	SEV_SOMEIP_ERROR
Parent Container	XfrmSecurityEventRefs
Upstream mapping defined	–
Description	The processing of a SOME/IP message resulted in a return not equal to E_OK.
Multiplicity	0..1

▽

△

Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[SWS_Xfrm_CONSTR_09097] SEV_SOMEIP_ERROR configurable only by SOME/IP transformer

Upstream requirements: SRS_Xfrm_00004

⌈The security event SEV_SOMEIP_ERROR shall only be configurable by the SOME/IP transformer.⌋

10.2.3 XfrmSignal

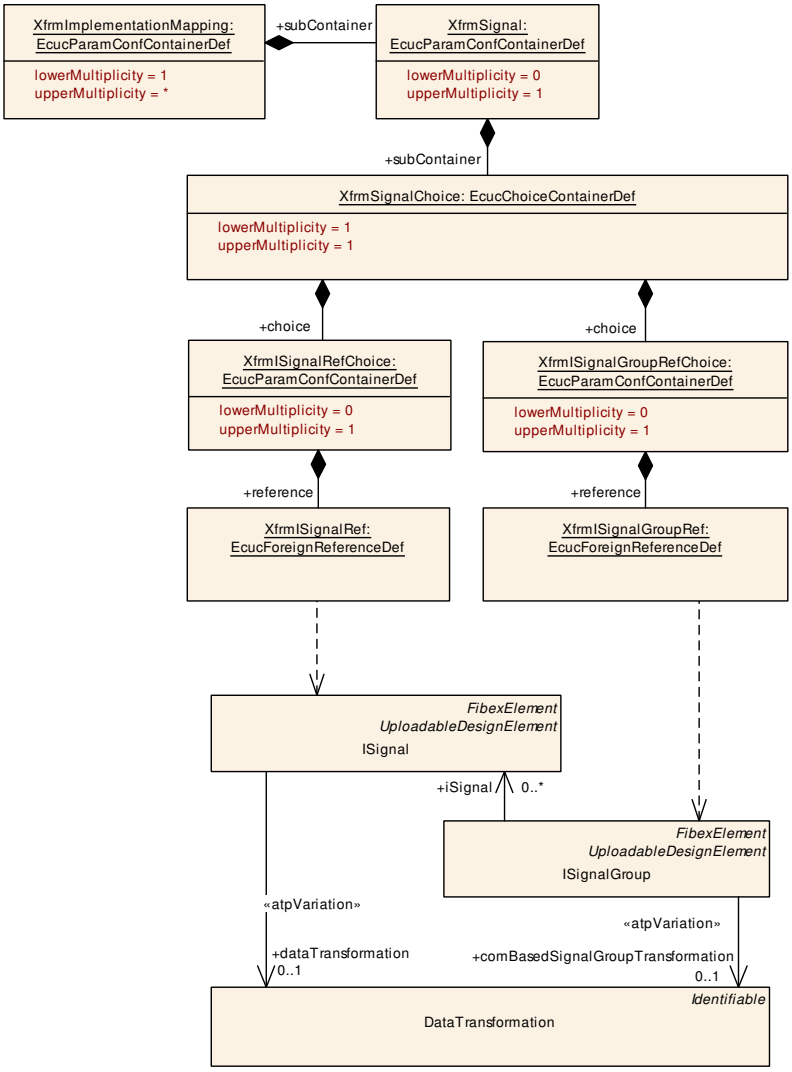

Figure 10.2: AR_EcucDef_XfrmSignal

[ECUC_Xfrm_00002] Definition of EcucParamConfContainerDef XfrmSignal ⌈

Container Name	XfrmSignal
Parent Container	XfrmImplementationMapping
Upstream mapping defined	–
Description	Reference to the signal in the system description that transports the transformed data.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	Multiplicity	Description
XfrmSignalChoice	1	Choice whether an ISignal or an ISignalGroup shall be referenced.

⌋

[ECUC_Xfrm_00006] Definition of EcucChoiceContainerDef XfrmSignalChoice ⌈

Choice Container Name	XfrmSignalChoice
Parent Container	XfrmSignal
Upstream mapping defined	–
Description	Choice whether an ISignal or an ISignalGroup shall be referenced.
Multiplicity	1
Pre-Compile only	true

No Included Parameters

Container Choices		
Container Name	Multiplicity	Description
XfrmISignalGroupRefChoice	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.

⌋

[ECUC_Xfrm_00009] Definition of EcucParamConfContainerDef XfrmISignalGroupRefChoice ⌈

Container Name	XfrmISignalGroupRefChoice
Parent Container	XfrmSignalChoice
Upstream mapping defined	–
Description	Reference to the ISignalGroup in the system description that transports the transformed data.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
XfrmISignalGroupRef	1	[ECUC_Xfrm_00010]

No Included Containers

⌋

[ECUC_Xfrm_00010] Definition of EcucForeignReferenceDef XfrmISignalGroupRef ⌈

Parameter Name	XfrmISignalGroupRef
Parent Container	XfrmISignalGroupRefChoice
Upstream mapping defined	[3, CP_TPS_SystemTemplate]
Description	Reference to the ISignalGroup in the system description that transports the transformed data.
Multiplicity	1
Type	Foreign reference to I-SIGNAL-GROUP
Pre-Compile only	true

⌋

[ECUC_Xfrm_00007] Definition of EcucParamConfContainerDef XfrmISignalRefChoice ⌈

Container Name	XfrmISignalRefChoice
Parent Container	XfrmSignalChoice
Upstream mapping defined	–
Description	Reference to the ISignal in the system description that transports the transformed data.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
XfrmISignalRef	1	[ECUC_Xfrm_00008]

No Included Containers

⌋

[ECUC_Xfrm_00008] Definition of EcucForeignReferenceDef XfrmISignalRef ⌈

Parameter Name	XfrmISignalRef
Parent Container	XfrmISignalRefChoice
Upstream mapping defined	[3, CP_TPS_SystemTemplate]
Description	Reference to the ISignal in the system description that transports the transformed data.
Multiplicity	1

▽

△

Type	Foreign reference to I-SIGNAL
Pre-Compile only	true

⌋

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.

Enumeration	**ArrayImplPolicyEnum**
Note	This meta-class provides values to configure the implementation of the payload part of an array.
Aggregated by	ImplementationDataTypeElement.arrayImplPolicy
Literal	***Description***
payloadAsArray	This configuration demands the implementation of the payload as an array. **Tags:** atp.EnumerationLiteralIndex=0
payloadAsPointerTo Array	This configuration demands the implementation of the payload as a pointer to an array. **Tags:** atp.EnumerationLiteralIndex=1

Table A.1: ArrayImplPolicyEnum

Class	***AtomicSwComponentType*** (abstract)			
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*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Subclasses	ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponent Type, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponent Type, ServiceSwComponentType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
internalBehavior	SwcInternalBehavior	0..1	aggr	The `SwcInternalBehavior`s owned by an `AtomicSwComponentType` can be located in a different physical file. Therefore the aggregation is <<atp Splitable>>. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=internalBehavior, internalBehavior.short Name, internalBehavior.variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the `SymbolProps` for the `AtomicSwComponentType`. **Tags:** atp.Splitkey=symbolProps, symbolProps.short Name

Table A.2: AtomicSwComponentType

Class	**BswModuleEntry**
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 This Class is only used by the AUTOSAR Classic Platform.

▽

△

Class	**BswModuleEntry**			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=45
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	0..1	attr	The type of call associated with this service. **Tags:** xml.sequenceOffset=25
execution Context	BswExecutionContext	0..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
function Prototype Emitter	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	0..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	0..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:** atp.Splitkey=returnType, returnType.shortName 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
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

▽

△

Class	**BswModuleEntry**			
swServiceImpl Policy	SwServiceImplPolicy Enum	0..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.3: BswModuleEntry

Class	**ClientServerInterface**			
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*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
operation	ClientServerOperation	*	aggr	`ClientServerOperation`(s) of this `ClientServerInterface`. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=operation, operation.shortName, operation.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime This Attribute is only used by the AUTOSAR Classic Platform.
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface. **Tags:** atp.Splitkey=possibleError, possibleError.shortName

Table A.4: ClientServerInterface

Class	**ClientServerOperation**			
Note	An operation declared within the scope of a client/server interface.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationInterface.command, *AtpClassifier*.atpFeature, ClientServerInterface.operation, DiagnosticDataElementInterface.read, DiagnosticDataIdentifierInterface.read, DiagnosticDataIdentifierInterface.write, DiagnosticExtendedDataRecordInterface.provide, DiagnosticRoutineInterface.requestResult, DiagnosticRoutineInterface.start, DiagnosticRoutineInterface.stop, ServiceInterface.method			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this `ClientServerOperation`. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime

▽

△

Class	**ClientServerOperation**			
diagArgIntegrity	Boolean	0..1	attr	This attribute shall only be used in the implementation of diagnostic routines to support the case where input and output arguments are allocated in a shared buffer and might unintentionally overwrite input arguments by tentative write operations to output arguments. This situation can happen during sliced execution or while output parameters are arrays (call by reference). The value true means that the `ClientServerOperation` is aware of the usage of a shared buffer and takes precautions to avoid unintentional overwrite of input arguments. If the attribute does not exist or is set to false the `ClientServerOperation` does not have to consider the usage of a shared buffer. This Attribute is only used by the AUTOSAR Classic Platform.
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation. **Tags:** atp.Splitkey=possibleError This Attribute is only used by the AUTOSAR Classic Platform.

Table A.5: ClientServerOperation

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

Table A.6: ClientServerToSignalMapping

Class	**DataPrototypeMapping**			
Note	Defines the mapping of two particular `VariableDataPrototype`s, `ParameterDataPrototype`s or `ArgumentDataPrototype`s with non-equal `shortName`s, non-equal structure (specific condition is described by [constr_1187]), and/or non-equal semantic (resolution or range) in context of two different `SenderReceiverInterface`, `NvDataInterface` or `ParameterInterface` or Operations. If the semantic is unequal, the following rules apply: The `textTableMapping` is only applicable if the referred `DataPrototype`s are typed by `AutosarDataType` referring to `CompuMethod`s of `category` `TEXTTABLE`, `SCALE_LINEAR_AND_TEXTTABLE` or `BITFIELD_TEXTTABLE`. In the case that the `DataPrototype`s are typed by `AutosarDataType` either referring to `CompuMethod`s 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 `Unit`s and the `CompuRationalCoeffs` of a `compuInternalToPhys` of the referred `CompuMethod`s.			
Base	*ARObject*			
Aggregated by	ClientServerOperationMapping.argumentMapping, VariableAndParameterInterfaceMapping.dataMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
firstData Prototype	AutosarDataPrototype	0..1	ref	First to be mapped DataPrototype in context of a Sender ReceiverInterface, NvDataInterface, ParameterInterface or Operation. **Tags:** atp.Splitkey=firstDataPrototype
firstToSecond Data Transformation	DataTransformation	0..1	ref	This reference defines the need to execute the Data Transformation <Mip>_<transformerId> functions of the transformation chain when communicating from the Data PrototypeMapping.firstDataPrototype to the Data PrototypeMapping.secondDataPrototype. This reference also specifies the reverse Data Transformation <Mip>_Inv_<transformerId> functions of the transformation chain (i.e. from the DataPrototype Mapping.secondDataPrototype to the DataPrototype Mapping.firstDataPrototype) if the referenced Data Transformation is symmetric, i.e. attribute Data Transformation.dataTransformationKind is set to symmetric. **Tags:** atp.Splitkey=firstToSecondDataTransformation
secondData Prototype	AutosarDataPrototype	0..1	ref	Second to be mapped DataPrototype in context of a SenderReceiverInterface, NvDataInterface, Parameter Interface or Operation. **Tags:** atp.Splitkey=secondDataPrototype
secondToFirst Data Transformation	DataTransformation	0..1	ref	This defines the need to execute the reverse Data Transformation <Mip>_Inv_<transformerId> functions of the transformation chain when communicating from the DataPrototypeMapping.secondDataPrototype to the Data PrototypeMapping.firstDataPrototype. **Tags:** atp.Splitkey=secondToFirstDataTransformation
subElement Mapping	SubElementMapping	*	aggr	This represents the owned SubelementMapping. **Tags:** atp.Splitkey=subElementMapping
textTable Mapping	TextTableMapping	*	aggr	Applied TextTableMapping(s) **Tags:** atp.Splitkey=textTableMapping

Table A.7: DataPrototypeMapping

Class	**DataTransformation**
Note	A DataTransformation represents a transformer chain. It is an ordered list of transformers.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*
Aggregated by	DataTransformationSet.dataTransformation

▽

△

Class	**DataTransformation**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
data Transformation Kind	DataTransformationKind Enum	0..1	attr	This attribute controls the kind of DataTransformation to be applied.
executeDespite Data Unavailability	Boolean	0..1	attr	Specifies whether the transformer chain is executed even if no input data are available.
transformer Chain (ordered)	Transformation Technology	*	ref	This attribute represents the definition of a chain of transformers that are supposed to be executed according to the order of being referenced from DataTransformation.

Table A.8: DataTransformation

Class	**DataTransformationSet**			
Note	This element is the system wide container of DataTransformations which represent transformer chains. **Tags:** atp.recommendedPackage=DataTransformationSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
data Transformation	DataTransformation	*	aggr	This container consists of all transformer chains which can be used for transformation of data communication. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=dataTransformation, data Transformation.shortName, dataTransformation.variation Point.shortLabel vh.latestBindingTime=codeGenerationTime
transformation Technology	Transformation Technology	*	aggr	Transformer that is used in a transformer chain for transformation of data communication. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=transformationTechnology, transformation Technology.shortName, transformation Technology.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table A.9: DataTransformationSet

Class	***IPdu*** (abstract)			
Note	The IPdu (Interaction Layer Protocol Data Unit) element is used to sum up all Pdus that are routed by the PduR.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Pdu*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Subclasses	ContainerIPdu, DcmIPdu, GeneralPurposeIPdu, ISignalIPdu, J1939DcmIPdu, J1939ProtectedIPdu, MultiplexedIPdu, NPdu, SecuredIPdu, UserDefinedIPdu			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.10: IPdu

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

△

Class	**ISignal**			
length	UnlimitedInteger	0..1	attr	Size of the signal in bits. The size needs to be derived from the mapped VariableDataPrototype according to the mapping of primitive DataTypes to BaseTypes as used in the RTE. Indicates maximum size for dynamic length signals. The ISignal length of zero bits is allowed.
network Representation Props	SwDataDefProps	0..1	aggr	Specification of the actual network representation. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType. The optional baseType attributes "memAllignment" and "byteOrder" shall not be used. 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 "ComSignalDataInvalid Value" and the Data Semantics. **Tags:** atp.Splitkey=networkRepresentationProps
systemSignal	SystemSignal	0..1	ref	Reference to the System Signal that is supposed to be transmitted in the ISignal. **Tags:** atp.Splitkey=systemSignal
timeout Substitution Value	ValueSpecification	0..1	aggr	Defines and enables the ComTimeoutSubstituition for this ISignal. **Tags:** atp.Splitkey=timeoutSubstitutionValue
transformation ISignalProps	TransformationISignal Props	*	aggr	A transformer chain consists of an ordered list of transformers. The ISignal specific configuration properties for each transformer are defined in the TransformationISignalProps class. The transformer configuration properties that are common for all ISignals are described in the TransformationTechnology class. **Tags:** atp.Splitkey=transformationISignalProps

Table A.11: ISignal

Class	**ISignalGroup**			
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=ISignalGroups			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**ISignalGroup**			
comBased SignalGroup Transformation	DataTransformation	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignalGroup based on the COMBasedTransformer approach. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=comBasedSignalGroupTransformation, comBasedSignalGroupTransformation.dataTransformation, comBasedSignalGroupTransformation.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime
iSignal	ISignal	*	ref	Reference to a set of ISignals that shall always be kept together. **Tags:** atp.Splitkey=iSignal
systemSignal Group	SystemSignalGroup	0..1	ref	Reference to the SystemSignalGroup that is defined on VFB level and that is supposed to be transmitted in the ISignalGroup. **Tags:** atp.Splitkey=systemSignalGroup
transformation ISignalProps	TransformationISignal Props	*	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 ISignal Groups are described in the TransformationTechnology class. **Tags:** atp.Splitkey=transformationISignalProps

Table A.12: ISignalGroup

Class	**ISignalToIPduMapping**			
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*			
Aggregated by	ISignalIPdu.iSignalToPduMapping, NmPdu.iSignalToIPduMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
iSignal	ISignal	0..1	ref	Reference to a ISignal that is mapped into the ISignal IPdu. 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. **Tags:** atp.Splitkey=iSignal
iSignalGroup	ISignalGroup	0..1	ref	Reference to an ISignalGroup that is mapped into the SignalIPdu. If an ISignalToIPduMapping for an ISignal Group is defined, only the UpdateIndicationBitPosition and the transferProperty is relevant. The startPosition and the packingByteOrder shall be ignored. 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. **Tags:** atp.Splitkey=iSignalGroup

▽

△

Class	**ISignalToIPduMapping**			
packingByte Order	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.
startPosition	UnlimitedInteger	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	Defines how the referenced ISignal contributes to the send triggering of the ISignalIPdu.
update IndicationBit Position	UnlimitedInteger	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 updateIndicationBit Position is linked to the value of the attribute packingByte Order because the method of finding the bit position is different for the values mostSignificantByteFirst and most SignificantByteLast. This means that if the value of packingByteOrder is changed while the value of update IndicationBitPosition remains unchanged the exact bit position of updateIndicationBitPosition within the enclosing 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.13: ISignalToIPduMapping

Class	**ImplementationDataType**
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*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Autosar DataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	**ImplementationDataType**			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to true, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional.
subElement (ordered)	ImplementationData TypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of `ImplementationDataTypeElement` 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 Implementation DataType. **Tags:** atp.Splitkey=symbolProps, symbolProps.short Name
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table A.14: ImplementationDataType

Class	**ImplementationDataTypeElement**			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further `subElement`s or it is further defined via its `swDataDefProps`. There are several use cases within the system of `ImplementationDataType`s for such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ImplementationDataType.subElement, ImplementationDataTypeElement.sub Element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
arrayImplPolicy	ArrayImplPolicyEnum	0..1	attr	This attribute controls the implementation of the payload of an array. It shall only be used if the enclosing `ImplementationDataType` constitutes an array.
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this `ImplementationDataTypeElement` represents the type of each single array element. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

▽

△

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

Table A.15: ImplementationDataTypeElement

Class	**NvBlockSwComponentType**			
Note	The NvBlockSwComponentType defines non volatile data which data can be shared between Sw ComponentPrototypes. The non volatile data of the NvBlockSwComponentType are accessible via provided and required ports. **Tags:** atp.recommendedPackage=SwComponentTypes This Class is only used by the AUTOSAR Classic Platform.			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Collectable Element*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
bulkNvData Descriptor	BulkNvDataDescriptor	*	aggr	This aggregation formally defines the bulk Nv Blocks that are provided to the application software by the enclosing NvBlockSwComponentType. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=bulkNvDataDescriptor, bulkNvData Descriptor.shortName, bulkNvDataDescriptor.variation Point.shortLabel vh.latestBindingTime=preCompileTime
nvBlock Descriptor	NvBlockDescriptor	*	aggr	Specification of the properties of exactly one NVRAM Block. **Stereotypes:** atpVariation **Tags:** atp.Splitkey=nvBlockDescriptor, nvBlockDescriptor.short Name, nvBlockDescriptor.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.16: NvBlockSwComponentType

Class	**PPortPrototype**			
Note	Component port providing a certain port interface.			
Base	*ARObject*, *AbstractProvidedPortPrototype*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *PortPrototype*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
provided Interface	PortInterface	0..1	tref	The interface that this port provides. **Stereotypes:** isOfType **Tags:** atp.Splitkey=providedInterface

Table A.17: PPortPrototype

Class	***PortInterfaceMapping*** (abstract)			
Note	Specifies one `PortInterfaceMapping` to support the connection of Ports typed by two different `PortInterface`s with PortInterface elements having unequal names and/or unequal semantic (resolution or range).			
Base	*ARObject*, *AtpBlueprint*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ClientServerInterfaceMapping, ModeInterfaceMapping, TriggerInterfaceMapping, VariableAndParameter InterfaceMapping			
Aggregated by	PortInterfaceMappingSet.portInterfaceMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.18: PortInterfaceMapping

Class	***PortPrototype*** (abstract)			
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*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Aggregated by	*AtpClassifier*.atpFeature, *SwComponentType*.port			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this `PortPrototype` with respect to client/ server communication. **Tags:** atp.Splitkey=clientServerAnnotation
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port. **Tags:** atp.Splitkey=delegatedPortAnnotation
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port. **Tags:** atp.Splitkey=ioHwAbstractionServerAnnotation
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port. **Tags:** atp.Splitkey=modePortAnnotation
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port. **Tags:** atp.Splitkey=nvDataPortAnnotation
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port. **Tags:** atp.Splitkey=parameterPortAnnotation

△

Class	***PortPrototype*** (abstract)			
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication. **Tags:** atp.Splitkey=senderReceiverAnnotation
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port. **Tags:** atp.Splitkey=triggerPortAnnotation

Table A.19: PortPrototype

Class	***Referrable*** (abstract)			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** atp.Splitkey=shortNameFragment xml.sequenceOffset=-90

Table A.20: Referrable

Class	**SenderRecArrayElementMapping**			
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*			
Aggregated by	SenderRecArrayTypeMapping.arrayElementMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
complexType Mapping	SenderRecComposite TypeMapping	0..1	aggr	This aggregation will be used if the element is composite. **Tags:** atp.Splitkey=complexTypeMapping

▽

△

Class	**SenderRecArrayElementMapping**			
indexedArray Element	IndexedArrayElement	0..1	aggr	Reference to an indexed array element in the context of the dataElement or in the context of a composite element. **Tags:** atp.Splitkey=indexedArrayElement
systemSignal	SystemSignal	0..1	ref	Reference to the system signal used to carry the primitive ApplicationArrayElement. **Tags:** atp.Splitkey=systemSignal

Table A.21: SenderRecArrayElementMapping

Class	**SenderRecRecordElementMapping**			
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 application RecordElement 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*			
Aggregated by	SenderRecRecordTypeMapping.recordElementMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
application RecordElement	ApplicationRecord Element	0..1	ref	Reference to an ApplicationRecordElement in the context of the dataElement or in the context of a composite element. **Tags:** atp.Splitkey=applicationRecordElement
complexType Mapping	SenderRecComposite TypeMapping	0..1	aggr	This aggregation will be used if the element is composite. **Tags:** atp.Splitkey=complexTypeMapping
implementation RecordElement	ImplementationData TypeElement	0..1	ref	Reference to an ImplementationRecordElement in the context of the dataElement or in the context of a composite element. **Tags:** atp.Splitkey=implementationRecordElement
senderToSignal TextTable Mapping	TextTableMapping	0..1	aggr	This mapping allows for the text-table translation between the sending DataPrototype that is defined in the Port Prototype and the physicalProps defined for the System Signal. **Tags:** atp.Splitkey=senderToSignalTextTableMapping
signalTo ReceiverText TableMapping	TextTableMapping	0..1	aggr	This mapping allows for the text-table translation between the physicalProps defined for the SystemSignal and a receiving DataPrototype that is defined in the Port Prototype. **Tags:** atp.Splitkey=signalToReceiverTextTableMapping
systemSignal	SystemSignal	0..1	ref	Reference to the system signal used to carry the primitive ApplicationRecordElement. **Tags:** atp.Splitkey=systemSignal

Table A.22: SenderRecRecordElementMapping

Class	**SenderReceiverInterface**			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataElement	VariableDataPrototype	*	aggr	The data elements of this `SenderReceiverInterface`. **Tags:** atp.Splitkey=dataElement, dataElement.shortName
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement **Tags:** atp.Splitkey=invalidationPolicy
metaDataItemSet	MetaDataItemSet	*	aggr	This aggregation defines fixed sets of meta-data items associated with `dataElement`s of the enclosing `SenderReceiverInterface` **Tags:** atp.Splitkey=metaDataItemSet

Table A.23: SenderReceiverInterface

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

Table A.24: SenderReceiverToSignalMapping

Class	**SwComponentPrototype**			
Note	Role of a software component within a composition.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, CompositionSwComponentType.component			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**SwComponentPrototype**			
type	SwComponentType	0..1	tref	Type of the instance. **Stereotypes:** isOfType **Tags:** atp.Splitkey=type

Table A.25: SwComponentPrototype

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

Table A.26: SystemSignal

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

Table A.27: TransformationTechnology

Class	Trigger			
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*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.releasedTrigger, BswModuleDescription.requiredTrigger, ServiceInterface.trigger, TriggerInterface.trigger			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers. This Attribute is only used by the AUTOSAR Classic Platform.
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger. **Tags:** atp.Splitkey=triggerPeriod This Attribute is only used by the AUTOSAR Classic Platform.

Table A.28: Trigger

Class	TriggerInterface			
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*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
trigger	Trigger	*	aggr	The Trigger of this trigger interface. **Tags:** atp.Splitkey=trigger, trigger.shortName

Table A.29: TriggerInterface

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

Table A.30: TriggerToSignalMapping

Class	**VariableDataPrototype**			
Note	A `VariableDataPrototype` represents a formalized generic piece of information that is typically mutable by the application software layer. `VariableDataPrototype` is used in various contexts and the specific context gives the otherwise generic `VariableDataPrototype` a dedicated semantics.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Aggregated by	ApplicationInterface.indication, *AtpClassifier*.atpFeature, BswInternalBehavior.arTypedPerInstance Memory, BswModuleDescription.providedData, BswModuleDescription.requiredData, BulkNvData Descriptor.bulkNvBlock, DiagnosticSovdAccessArgument.contentObject, *InternalBehavior*.staticMemory, NvBlockDescriptor.ramBlock, NvDataInterface.nvData, SenderReceiverInterface.dataElement, Service Interface.event, SwcInternalBehavior.arTypedPerInstanceMemory, SwcInternalBehavior.explicitInter RunnableVariable, SwcInternalBehavior.implicitInterRunnableVariable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype **Tags:** atp.Splitkey=initValue

Table A.31: VariableDataPrototype

B Change history of AUTOSAR traceable items

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

B.1 Traceable item history of this document according to AUTOSAR Release R23-11

B.1.1 Added Specification Items in R23-11

none

B.1.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Xfrm_00031]	Errors of serializer transformers
[SWS_Xfrm_00032]	Errors of safety transformers
[SWS_Xfrm_00033]	Errors of security transformers
[SWS_Xfrm_00036]	Definition of API function <Mip>_<transformerId>
[SWS_Xfrm_00038]	Definition of API function <Mip>_<transformerId>
[SWS_Xfrm_00040]	Definition of API function <Mip>_<transformerId>
[SWS_Xfrm_00042]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_00044]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_00046]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_00050]	Errors of custom transformers
[SWS_Xfrm_00060]	Definition of datatype {Mip}_ConfigType
[SWS_Xfrm_00070]	Extended Production Errors of transformer
[SWS_Xfrm_00102]	Definition of API function <Mip>_<transformerId>
[SWS_Xfrm_00104]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_91001]	Definition of imported datatypes of module Xfrm
[SWS_Xfrm_91002]	Definition of API function <Mip>_ExtractProtocolHeaderFields

Table B.1: Changed Specification Items in R23-11

B.1.3 Deleted Specification Items in R23-11

none

B.2 Traceable item history of this document according to AUTOSAR Release R24-11

B.2.1 Added Specification Items in R24-11

none

B.2.2 Changed Specification Items in R24-11

Number	Heading
[ECUC_Xfrm_00014]	Definition of EcucModuleDef Xfrm
[SWS_Xfrm_00058]	Definition of API function <Mip>_Init
[SWS_Xfrm_00059]	Definition of API function <Mip>_DeInit

Table B.2: Changed Specification Items in R24-11

B.2.3 Deleted Specification Items in R24-11

none

B.2.4 Added Constraints in R24-11

none

B.2.5 Changed Constraints in R24-11

none

B.2.6 Deleted Constraints in R24-11

none

B.3 Traceable item history of this document according to AUTOSAR Release R25-11

B.3.1 Added Specification Items in R25-11

Number	Heading
[SWS_Xfrm_02000]	Variable prototype parts of [SWS_Xfrm_00036]
[SWS_Xfrm_02001]	Variable prototype parts of [SWS_Xfrm_00038]
[SWS_Xfrm_02002]	Variable prototype parts of [SWS_Xfrm_00102]
[SWS_Xfrm_02003]	Variable prototype parts of [SWS_Xfrm_00040]
[SWS_Xfrm_02004]	Variable prototype parts of [SWS_Xfrm_00042]
[SWS_Xfrm_02005]	Variable prototype parts of [SWS_Xfrm_00044]
[SWS_Xfrm_02006]	Variable prototype parts of [SWS_Xfrm_00104]
[SWS_Xfrm_02007]	Variable prototype parts of [SWS_Xfrm_00046]
[SWS_Xfrm_02008]	Variable prototype parts of [SWS_Xfrm_00058]
[SWS_Xfrm_02009]	Variable prototype parts of [SWS_Xfrm_00059]
[SWS_Xfrm_02010]	Variable prototype parts of [SWS_Xfrm_00057]

Table B.3: Added Specification Items in R25-11

B.3.2 Changed Specification Items in R25-11

Number	Heading
[SWS_Xfrm_00031]	Errors of serializer transformers
[SWS_Xfrm_00042]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_00044]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_00062]	

Table B.4: Changed Specification Items in R25-11

B.3.3 Deleted Specification Items in R25-11

none

B.3.4 Added Constraints in R25-11

none

B.3.5 Changed Constraints in R25-11

none

B.3.6 Deleted Constraints in R25-11

none

B.4 Traceable item history of this document according to AUTOSAR Release R26-11

B.4.1 Added Specification Items in R26-11

Number	Heading
[ECUC_Xfrm_00023]	Definition of EcucBooleanParamDef XfrmEnableSecurityEventReporting
[ECUC_Xfrm_00024]	Definition of EcucParamConfContainerDef XfrmSecurityEventRefs
[ECUC_Xfrm_00025]	Definition of EcucReferenceDef SEV_SOMEIP_ERROR

Table B.5: Added Specification Items in R26-11

B.4.2 Changed Specification Items in R26-11

Number	Heading
[ECUC_Xfrm_00001]	Definition of EcucParamConfContainerDef XfrmImplementationMapping
[ECUC_Xfrm_00012]	Definition of EcucParamConfContainerDef XfrmGeneral
[SWS_Xfrm_00038]	Definition of API function <Mip>_<transformerId>
[SWS_Xfrm_00040]	Definition of API function <Mip>_<transformerId>
[SWS_Xfrm_00044]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_00046]	Definition of API function <Mip>_Inv_<transformerId>
[SWS_Xfrm_02001]	Variable prototype parts of [SWS_Xfrm_00038]
[SWS_Xfrm_02003]	Variable prototype parts of [SWS_Xfrm_00040]
[SWS_Xfrm_02005]	Variable prototype parts of [SWS_Xfrm_00044]
[SWS_Xfrm_02007]	Variable prototype parts of [SWS_Xfrm_00046]
[SWS_Xfrm_91001]	Definition of imported datatypes of module Xfrm

Table B.6: Changed Specification Items in R26-11

B.4.3 Deleted Specification Items in R26-11

Number	Heading
[ECUC_Xfrm_00015]	Definition of EcucReferenceDef XFRM_E_MALFORMED_MESSAGE
[ECUC_Xfrm_00016]	Definition of EcucParamConfContainerDef XfrmDemEventParameterRefs
[ECUC_Xfrm_00019]	Definition of EcucBooleanParamDef XfrmVersionInfoApi
[ECUC_Xfrm_00021]	Definition of EcucForeignReferenceDef XfrmCSTransactionHandle ImplementationDataTypeRef
[SWS_Xfrm_00057]	Definition of API function <Mip>_GetVersionInfo
[SWS_Xfrm_00070]	Extended Production Errors of transformer
[SWS_Xfrm_00071]	
[SWS_Xfrm_02010]	Variable prototype parts of [SWS_Xfrm_00057]

Table B.7: Deleted Specification Items in R26-11

B.4.4 Added Constraints in R26-11

Number	Heading
[SWS_Xfrm_CONSTR_09097]	SEV_SOMEIP_ERROR configurable only by SOME/IP transformer

Table B.8: Added Constraints in R26-11

B.4.5 Changed Constraints in R26-11

none

B.4.6 Deleted Constraints in R26-11

none

C Not applicable requirements

[SWS_Xfrm_NA_00001]

Upstream requirements: SRS_BSW_00004, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00369, SRS_BSW_00375, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00403, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00425, SRS_BSW_00432, SRS_BSW_00461, SRS_BSW_00469, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00478, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00497

⌈These requirements are not applicable to this specification.⌋