| Document Title | Specification of Data Distribution Service Transformer |
|---|---|
| **Document Category** | Software specification (SWS) |
| **Document UID** | 1140 |
| **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 | • Remove GetVersionInfo API<br>• Minor corrections / clarifications / editorial changes |
| 2025-11-27 | R25-11 | 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 6
2 Acronyms, Abbreviations and Definitions 7
3 Related documentation 8
3.1 Input documents & related standards and norms 8
4 Constraints and assumptions 9
4.1 Limitations 9
5 Dependencies to other modules 10
5.1 File structure 10
6 Requirements Tracing 11
7 Functional specification 13
7.1 Transformer infrastructure header 15
7.2 Marshalling of Parameters and Data Structures 16
7.3 Unmarshalling of Parameters and Data Structures 17
7.4 Error Classification 18
7.4.1 Development Errors 18
7.4.2 Runtime Errors 18
7.4.3 Production Errors 19
8 API specification 20
8.1 Imported types 20
8.2 Type definitions 20
8.3 Function definitions 20
8.3.1 SenderReceiverInterface API 21
8.3.1.1 DdsXf_<transformerId> 21
8.3.1.2 DdsXf_Inv_<transformerId> 23
8.3.2 ClientServerInterface API 25
8.3.2.1 DdsXf_<transformerId> 25
8.3.2.2 DdsXf_Inv_<transformerId> 27
8.4 Scheduled functions 30
8.5 Expected interfaces 30
9 Sequence diagrams 31
10 Configuration specification 32
10.1 How to read this chapter 32
10.2 Containers and configuration parameters 32
10.3 Published Information 32
A Not applicable requirements 33

B Change history of AUTOSAR traceable items 34
B.1 Change History R25-11 . . . 34
B.1.1 Added Specification Items in R25-11 . . . 34
B.1.2 Changed Specification Items in R25-11 . . . 34
B.1.3 Deleted Specification Items in R25-11 . . . 34
B.1.4 Added Constraints in R25-11 . . . 34
B.1.5 Changed Constraints in R25-11 . . . 34
B.1.6 Deleted Constraints in R25-11 . . . 35
B.2 Change History R26-11 . . . 35
B.2.1 Added Specification Items in R26-11 . . . 35
B.2.2 Changed Specification Items in R26-11 . . . 35
B.2.3 Deleted Specification Items in R26-11 . . . 35
B.2.4 Added Constraints in R26-11 . . . 35
B.2.5 Changed Constraints in R26-11 . . . 35
B.2.6 Deleted Constraints in R26-11 . . . 35
C Referenced Meta Classes 36

# Known Limitations

None.

# 1 Introduction and functional overview

This specification describes the functionality, API, and the configuration of **Data Distribution Service Transformer** in the context of AUTOSAR Classic Platform.

The role of the DDS Transformer is to copy input data elements (`VariableDataPrototype`) and operations (`ClientServerOperation`s) from the application layer into a buffer provided by the RTE. And vice versa, the DDS transformer has to copy received data to data elements of the buffer provided by the application.

Those procedures are referred in this document as **Marshalling** and **Unmarshalling** of `VariableDataPrototype`s and `ClientServerOperation`s.

DDS Transformer is stateless, no status information needs to be stored.

The DDS Transformer, unlike SOME/IP [1], does not implement a network protocol. Its role is to encode and decode the information exchanged between the RTE and the Dds via a buffer, within the same ECU (see Figure 7.1).

# 2 Acronyms, Abbreviations and Definitions

The glossary below includes acronyms, abbreviations and definitions relevant to the DDS Transformer module that are not included in the [2, AUTOSAR TR Glossary].

| Abbreviation / Acronym: | Description: |
|---|---|
| Dds | Data Distribution Service Basic software module (i.e the DDS middleware implementation in AUTOSAR CP) |
| DdsXf | Data Distribution Service Transformer, DDS Transformer |

**Table 2.1: Acronyms and abbreviations used in the scope of this Document**

| Definition: | Description: |
|---|---|
| TransactionHandle | See SWS_Rte_08732 in [3] |

**Table 2.2: Definitions used in the scope of this Document**

# 3 Related documentation

## 3.1 Input documents & related standards and norms

[1] Specification of SOME/IP Transformer
AUTOSAR_CP_SWS_SOMEIPTransformer

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[4] General Specification of Transformers
AUTOSAR_CP_SWS_TransformerGeneral

[5] Requirements on Transformer
AUTOSAR_CP_RS_Transformer

[6] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[7] System Template
AUTOSAR_CP_TPS_SystemTemplate

[8] Specification of Data Distribution Service for Classic Platform
AUTOSAR_CP_SWS_DataDistributionService

[9] Specification of Platform Types for Classic Platform
AUTOSAR_CP_SWS_PlatformTypes

[10] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

# 4 Constraints and assumptions

The DDS Transformer can be used for all domain applications when SenderReceiverInterface or ClientServerInterface communication is used.

TriggerInterface typed data structure consists of an ISignal with length equal to zero and DdsXf provides an infrastructure header only for ClientServerInterface. For this reason, DDS Transformer does not support the TriggerInterface and DdsXf_ExtractProtocolHeaderFields function.

## 4.1 Limitations

For the DDS Transformer all general transformer limitations [4] apply.

# 5 Dependencies to other modules

The AUTOSAR RTE [3] has to exist to execute the transformer.

## 5.1 File structure

The source code file structure is defined in the [4].

# 6 Requirements Tracing

The following tables reference the requirements specified in [5] and in [6] and links to the fulfillment of these.

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_BSW_00337]** | Classification of development errors | [CP_SWS_DdsXf_00184] |
| **[SRS_BSW_00357]** | For success/failure of an API call a standard return type shall be defined | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00369]** | All AUTOSAR Basic Software Modules shall not return specific development error codes via the API | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00383]** | The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description | [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00385]** | List possible error notifications | [CP_SWS_DdsXf_00184] |
| **[SRS_BSW_00392]** | Parameters shall have a type | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00404]** | BSW Modules shall support post-build configuration | [CP_SWS_DdsXf_00183] |
| **[SRS_BSW_00417]** | Software which is not part of the SW-C shall report error events only after the Dem is fully operational. | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00422]** | Pre-de-bouncing of error status information is done within the Dem | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00432]** | Modules should have separate main processing functions for read/receive and write/transmit data path | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00462]** | All Standardized Autosar Interfaces shall have unique requirement Id / number | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00484]** | Input parameters of scalar and enum types shall be passed as a value. | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00485]** | Input parameters of structure type shall be passed as a reference to a constant structure | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_BSW_00486]** | Input parameters of array type shall be passed as a reference to the constant array base type | [CP_SWS_DdsXf_00138] |
| **[SRS_BSW_00494]** | ServiceInterface argument with a pointer datatype | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_Xfrm_00001]** | A transformer shall work on data given by the Rte | [CP_SWS_DdsXf_00264] [CP_SWS_DdsXf_00265] |
| **[SRS_Xfrm_00002]** | A transformer shall provide fixed interfaces | [CP_SWS_DdsXf_00139] [CP_SWS_DdsXf_00142] [CP_SWS_DdsXf_00145] [CP_SWS_DdsXf_00146] [CP_SWS_DdsXf_00147] [CP_SWS_DdsXf_00149] [CP_SWS_DdsXf_00152] [CP_SWS_DdsXf_00153] [CP_SWS_DdsXf_00161] [CP_SWS_DdsXf_00165] [CP_SWS_DdsXf_00166] [CP_SWS_DdsXf_00228] [CP_SWS_DdsXf_00231] [CP_SWS_DdsXf_00232] [CP_SWS_DdsXf_00266] [CP_SWS_DdsXf_91002] |
| **[SRS_Xfrm_00004]** | A transformer shall support error handling | [CP_SWS_DdsXf_00264] [CP_SWS_DdsXf_00265] |
| **[SRS_Xfrm_00007]** | A deserializer transformer shall support extraction of data | [CP_SWS_DdsXf_00144] |



△

| Requirement | Description | Satisfied by |
|---|---|---|
| **[SRS_Xfrm_00009]** | A fixed set of transformer classes shall exist | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00144] |
| **[SRS_Xfrm_00301]** | The DDS Transformer shall define a copy procedure of atomic, structured data elements and operations into a memory buffer | [CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00140] [CP_SWS_DdsXf_00141] [CP_SWS_DdsXf_00143] [CP_SWS_DdsXf_00148] [CP_SWS_DdsXf_00160] [CP_SWS_DdsXf_00163] [CP_SWS_DdsXf_00164] [CP_SWS_DdsXf_00168] [CP_SWS_DdsXf_00229] [CP_SWS_DdsXf_00728] [CP_SWS_DdsXf_00730] [CP_SWS_DdsXf_00731] [CP_SWS_DdsXf_00732] [CP_SWS_DdsXf_00738] [CP_SWS_DdsXf_00740] [CP_SWS_DdsXf_00741] [CP_SWS_DdsXf_00742] [CP_SWS_DdsXf_00751] |
| **[SRS_Xfrm_00305]** | The DDS Transformer shall support autonomous error reactions on the server side for client/server communication | [CP_SWS_DdsXf_00160] [CP_SWS_DdsXf_00161] [CP_SWS_DdsXf_00167] [CP_SWS_DdsXf_00169] [CP_SWS_DdsXf_00170] |

**Table 6.1: Requirements Tracing**

# 7 Functional specification

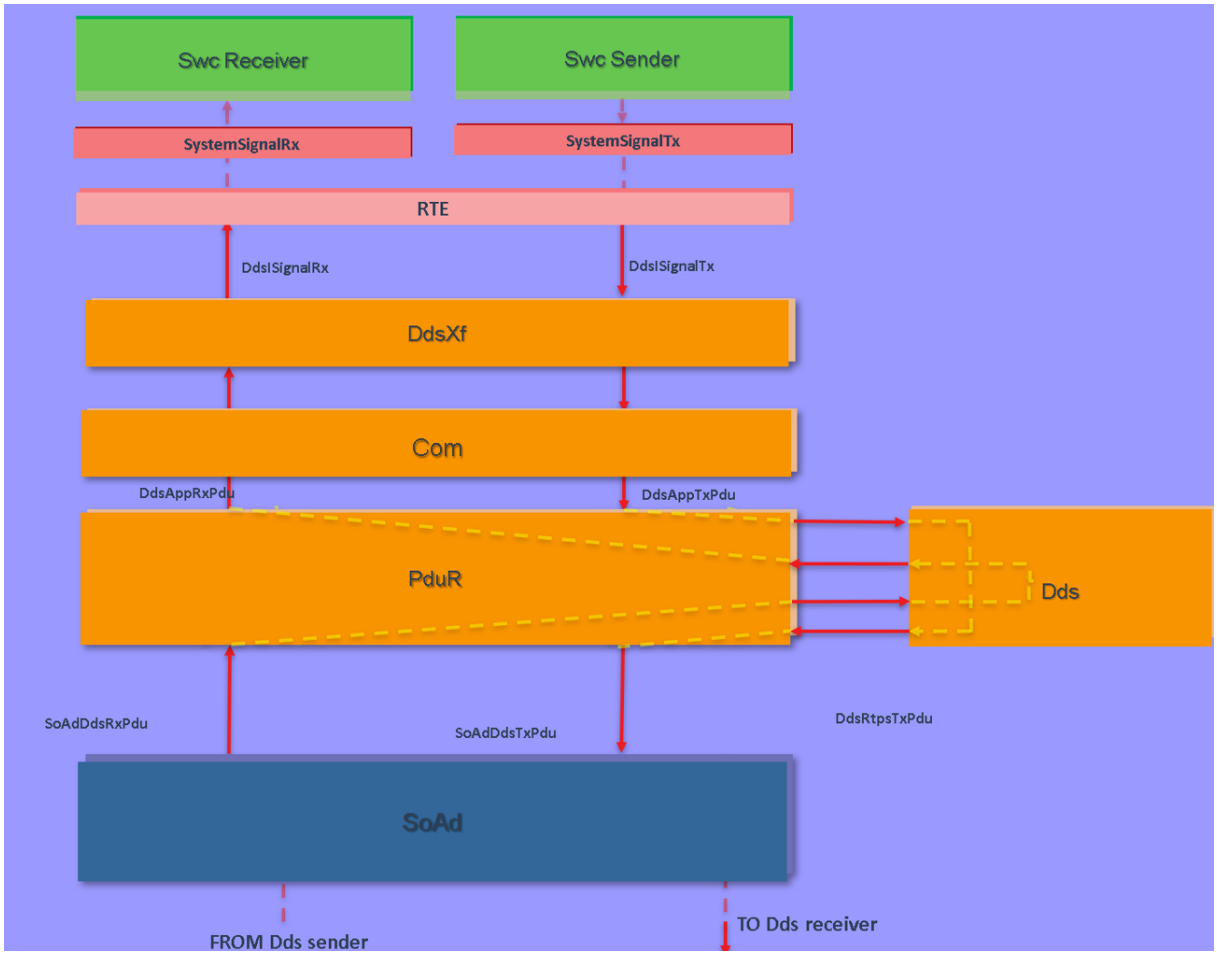


**Figure 7.1: Overview of DDS Transformer**

Figure 7.1 shows the AUTOSAR Classic Platform communication stack where the DDS Transformer is used together with the Dds in a `ClientServerInterface` application example on server side.

When a SW-C initiates an inter-ECU communication which is configured to be transformed, the SW-C hands the data over to the RTE. The RTE executes the configured transformer chain which contains the DDS Transformer.

First, the RTE executes the application sample through DdsXf, which creates a copy of the SW-C data (note: the data is copied into a memory buffer provided by the RTE) and sends it as a PDU to Dds. Subsequently, Dds extracts the data from the PDU, serializes it, and packages it into one or more DDS-RTPS message PDUs. Finally, the PDU is sent to the network stack (SoAd, TcpIp, etc.).

The DDS Transformer on the receiver side copies the data back from the received PDU into the original data structure in a symmetrical way. These are handed over to the receiving SW-C. From the SW-C's point of view, it is totally transparent whether data

are transformed or not. It provides also a subset of the transformer errors specified for this transformer class and supports only out-of-place buffer handling.

The DDS Transformer is a transformer of the class **Serializer**.

At most one transformer of each transformer class shall be allowed per transformer chain [4, Transformer Classes] and a Serializer class transformer shall map structured data type to a linear byte array, so that DDS Transformer can be only the first transformer in a chain. This property is formalized by [CP_SWS_DdsXf_00139], [CP_SWS_DdsXf_00146], [CP_SWS_DdsXf_00142], [CP_SWS_DdsXf_00232].

In addition, if the handling of safety and security is performed in scope of the Dds then no further transformers shall be defined in combination with DDS Transformer as stated by [constr_3822] in [7, SystemTemplate].

The DDS Transformer has no module specific EcuC because its whole configuration is based on the `DdsTransformationDescription` and `DdsTransformationISignalProps`.

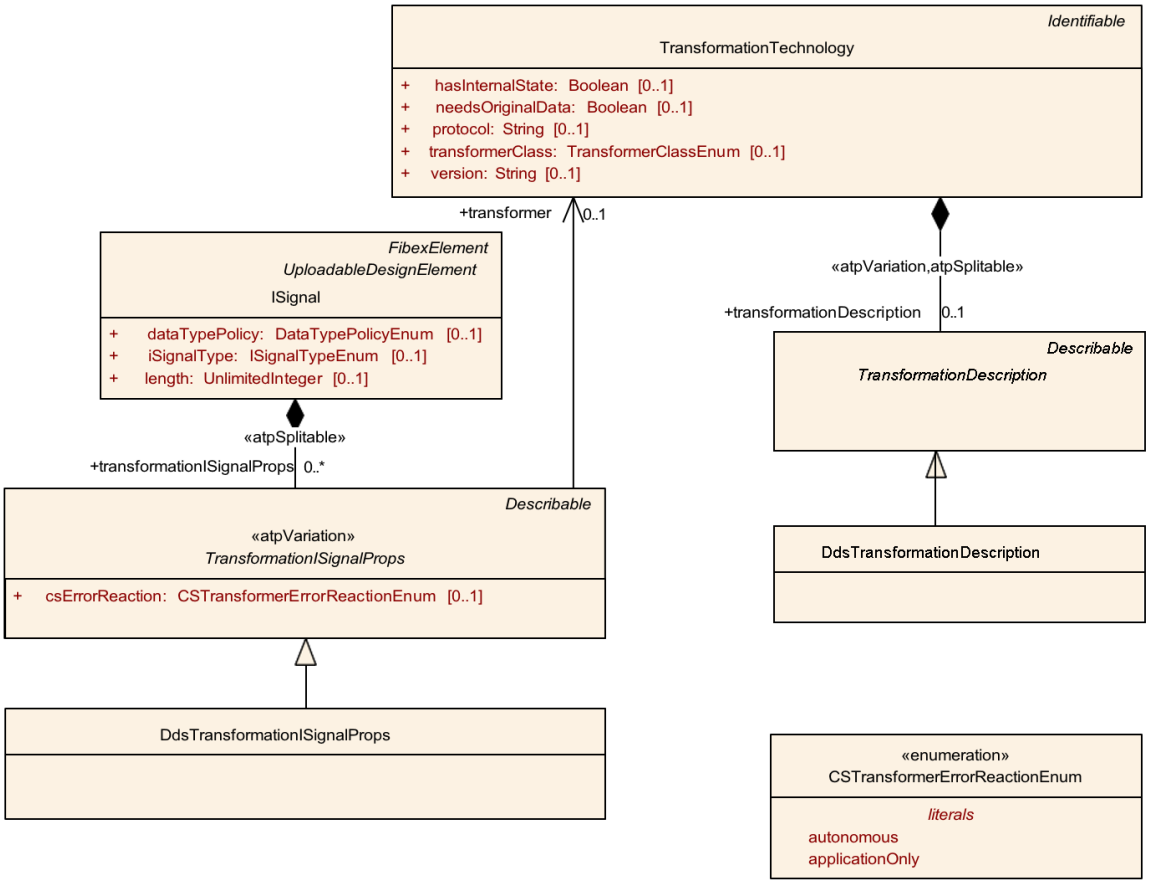


**Figure 7.2: DDS Transformer Specific Configuration**

| *Class* | **DdsTransformationDescription** | | | |
|---|---|---|---|---|
| *Note* | The DdsTransformationDescription is used to specify the DDS transformer specific attributes.<br>**Tags:** atp.Status=candidate | | | |
| *Base* | *ARObject*, *Describable*, *TransformationDescription* | | | |
| *Aggregated by* | TransformationTechnology.transformationDescription | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| – | – | – | – | – |

**Table 7.1: DdsTransformationDescription**

| *Class* | «atpVariation» **DdsTransformationISignalProps** | | | |
|---|---|---|---|---|
| *Note* | The class DdsTransformationISignalProps specifies ISignal specific configuration properties for the DDS transformer.<br>**Tags:** atp.Status=candidate | | | |
| *Base* | *ARObject*, *Describable*, *TransformationISignalProps* | | | |
| *Aggregated by* | ISignal.transformationISignalProps, ISignalGroup.transformationISignalProps | | | |
| *Attribute* | *Type* | *Mult.* | *Kind* | *Note* |
| – | – | – | – | – |

**Table 7.2: DdsTransformationISignalProps**

**[SWS_DdsXf_CONSTR_00151] DDS Transformer configuration** ⌈The DDS Transformer shall be configured according to [constr_3821] defined in [7]⌋

## 7.1 Transformer infrastructure header

DDS Transformer **infrastructure header** parameters are additional information the DDS Transformer adds into the provided buffer in `ClientServerInterface` context only.

This information is used by the Dds to support the Request-Response Method feature implemented by the DDS middleware as described in [8].

### [CP_SWS_DdsXf_00751] Infrastructure header definition

*Upstream requirements:* SRS_Xfrm_00301

⌈`DdsXf` infrastructure header shall be defined as follow:

- `requestId` [32 bit]
  - `clientId` [16 bit]
  - `sequenceCounter` [16 bit]
- `reserved` [24 bit]
- `returnValue` [8 bit]

⌋

**[CP_SWS_DdsXf_CONSTR_00735] reserved parameter of infrastructure header**
⌈The `reserved` parameter of the infrastructure header data structure shall be set to `0`.⌋

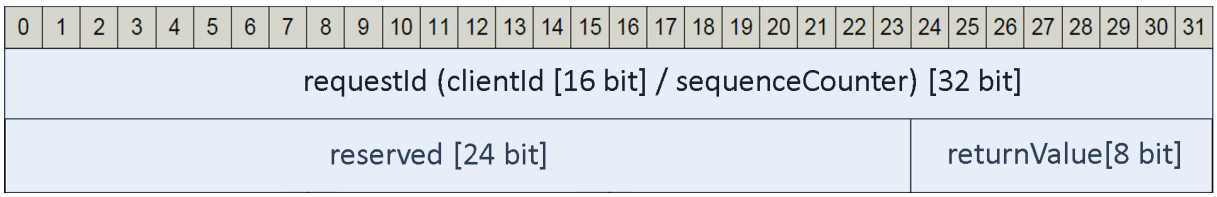


**Figure 7.3: Format of infrastructure header**

The complete memory layout of the transformed buffer is showed in Figure 7.4.

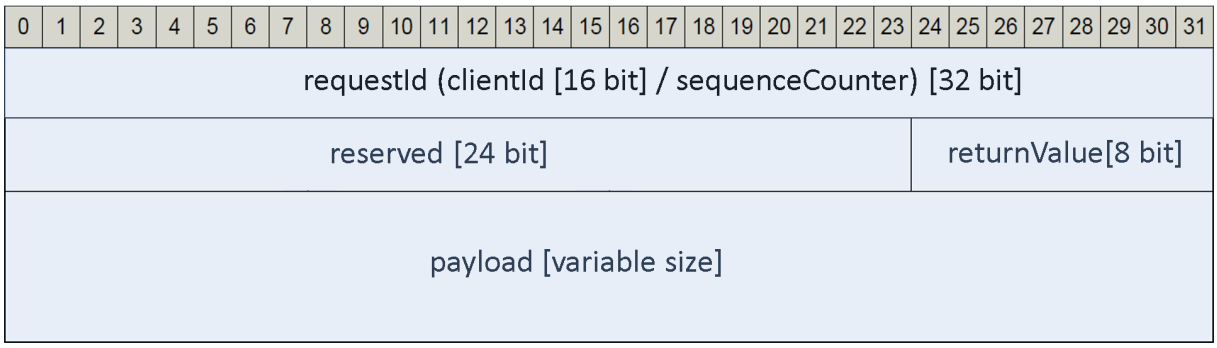


**Figure 7.4: The complete buffer**

## 7.2 Marshalling of Parameters and Data Structures

**[CP_SWS_DdsXf_00728] Marshalling of primitive data types**

*Upstream requirements:* SRS_Xfrm_00301

⌈The `DdsXf` shall perform a raw copy of the source AUTOSAR primitive data types into the provided memory buffer.⌋

**[CP_SWS_DdsXf_00730] Marshalling of ImplementationDataType of category ARRAY**

*Upstream requirements:* SRS_Xfrm_00301

⌈The `DdsXf` shall perform a raw copy of the source `ImplementationDataType` of category **ARRAY** into the provided memory buffer.⌋

**[CP_SWS_DdsXf_00731] Marshalling of ImplementationDataType of category STRUCTURE**

*Upstream requirements:* SRS_Xfrm_00301

⌈The `DdsXf` shall perform a raw copy of the source `ImplementationDataType` of category **STRUCTURE** into the provided memory buffer.⌋

**[CP_SWS_DdsXf_00732] Marshalling of ImplementationDataType of category TYPE_REFERENCE**

*Upstream requirements:* SRS_Xfrm_00301

⌈The DdsXf shall perform a raw copy of the source ImplementationDataType of category **TYPE_REFERENCE** into the provided memory buffer.⌋

**Note.** Data type defined according to [CP_SWS_DdsXf_00732] is also referred as "Redefinition Implementation Data Type" as stated by [3, chapter 5.3.4.2].

**[CP_SWS_DdsXf_CONSTR_00733] Marshalling of UNION data type** ⌈ImplementationDataType of category **UNION** is not managed by the Dds Transformer. The DDS Transformer BSW configuration validation shall fail in case a data type is typed with ImplementationDataType containing a union.⌋

**[CP_SWS_DdsXf_CONSTR_00734] Marshalling of POINTER data type** ⌈ImplementationDataType of category **POINTER** is not managed by the Dds. The DDS Transformer BSW configuration validation shall fail in case a data type is typed with ImplementationDataType containing a pointer.⌋

**Note**. The handling of any kind of **String** type (e.g fixed length, variable length, different encoding) is covered by [CP_SWS_DdsXf_00730], [CP_SWS_DdsXf_00740]. The DDS Transformer performs a raw copy of the parameters and it does not perform any check on the data structure consistency. Those kind of checks are up to the DDS middleware logic implemented in the Dds.

For details about Primitive AUTOSAR CP platform data types, Enumeration and ImplementationDataType refer to 8.2 [9], and 5.5.4, 5.3.4.2 [3].

## 7.3 Unmarshalling of Parameters and Data Structures

**[CP_SWS_DdsXf_00738] Unmarshalling of primitive data types**

*Upstream requirements:* SRS_Xfrm_00301

⌈The DdsXf shall retrieve an AUTOSAR primitive data type from a source memory buffer representing the same AUTOSAR primitive data type created according to [CP_SWS_DdsXf_00728]⌋

**[CP_SWS_DdsXf_00740] Unmarshalling of ImplementationDataType of category ARRAY**

*Upstream requirements:* SRS_Xfrm_00301

⌈The DdsXf shall retrieve an ImplementationDataType of category **ARRAY** from a source memory buffer created according to [CP_SWS_DdsXf_00730]⌋

**[CP_SWS_DdsXf_00741] Unmarshalling of ImplementationDataType of category STRUCTURE**

*Upstream requirements:* SRS_Xfrm_00301

⌈The `DdsXf` shall retrieve an `ImplementationDataType` of category **STRUCTURE** from a source buffer memory created according to [CP_SWS_DdsXf_00731]⌋

**[CP_SWS_DdsXf_00742] Unmarshalling of ImplementationDataType of category TYPE_REFERENCE**

*Upstream requirements:* SRS_Xfrm_00301

⌈The `DdsXf` shall retrieve an `ImplementationDataType` of category **TYPE_REFERENCE** from a source memory buffer created according to [CP_SWS_DdsXf_00732]⌋

## 7.4 Error Classification

Section "Error Handling" of the document [10] "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five 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

**[CP_SWS_DdsXf_00184] Definition of development errors in module DdsXf**

*Upstream requirements:* SRS_BSW_00337, SRS_BSW_00385

⌈

| *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. | DDSXF_E_UNINIT | 0x01 |
| Error code if an invalid configuration set was selected | DDSXF_E_INIT_FAILED | 0x02 |
| API service called with wrong parameter | DDSXF_E_PARAM | 0x03 |
| API service called with invalid pointer | DDSXF_E_PARAM_POINTER | 0x04 |

⌋

### 7.4.2 Runtime Errors

There are no runtime errors.

### 7.4.3 Production Errors

There are no production errors.

# 8 API specification

## 8.1 Imported types

There are no imported types from other modules beyond those specified in [4, SWS Transformer General].

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

**[CP_SWS_DdsXf_91002] Definition of imported datatypes of module DdsXf**

*Upstream requirements:* SRS_Xfrm_00002

⌈

| *Module* | *Header File* | *Imported Type* |
|---|---|---|
| Rte | SchM_<Mip>.h | Rte_Cs_TransactionHandleType |
| Std | Std_Types.h | Std_ReturnType |

⌋

## 8.2 Type definitions

**[CP_SWS_DdsXf_00183] Definition of datatype DdsXf_ConfigType**

*Upstream requirements:* SRS_BSW_00404

⌈

| | | |
|---|---|---|
| ***Name*** | DdsXf_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*** | DdsXf.h | |

⌋

## 8.3 Function definitions

This chapter contains all APIs provided by the module.

**[CP_SWS_DdsXf_00266] DDS Transformer shall define DdsTransformationDescription**

*Upstream requirements:* SRS_Xfrm_00002

⌈The DDS Transformer shall only provide functions for transformers where the `TransformationTechnology` aggregates a `DdsTransformationDescription` in the role `transformationDescription`.⌋

### 8.3.1 SenderReceiverInterface API

#### 8.3.1.1 DdsXf_<transformerId>

**[CP_SWS_DdsXf_00138] Definition of API function DdsXf_<transformerId>**

*Upstream requirements:* SRS_Xfrm_00301, SRS_Xfrm_00009, SRS_BSW_00494, SRS_BSW_00486, SRS_BSW_00485, SRS_BSW_00484, SRS_BSW_00462, SRS_BSW_00432, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00392, SRS_BSW_00369, SRS_BSW_00357

⌈

| | | |
|---|---|---|
| ***Service Name*** | DdsXf_<transformerId> | |
| ***Syntax*** | `uint8 DdsXf_<transformerId> ( uint8* buffer, uint32* bufferLength, <paramtype> dataElement )` | |
| ***Service ID [hex]*** | 0x03 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | dataElement | Data element which shall be transformed |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | buffer | Buffer allocated by the RTE, where the transformed data has to be stored by the transformer |
| | bufferLength | Used length of the buffer |
| ***Return value*** | uint8 | 0x00 (E_OK): Marshalling successful |
| ***Description*** | This function transforms a Sender/Receiver communication using the marshalling procedure of Dds. It takes the data element as input and outputs a uint8 array containing the marshalled data. The length of the marshalled 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*** | DdsXf.h | |

⌋

**[CP_SWS_DdsXf_00228] DdsXf_<transformerId> parameters definition**

*Upstream requirements:* SRS_Xfrm_00002

⌈In function `DdsXf_<transformerId>` defined in [CP_SWS_DdsXf_00138] the parameters:

- `paramtype`
- `type`

- transformerId

shall be defined according to [SWS_Xfrm_00036] in [4, TransformerGeneral].⌋

The function specified in [CP_SWS_DdsXf_00138] exists for each transformed SenderReceiverInterface communication which uses the Dds marshalling of VariableDataPrototypes procedure according to [CP_SWS_DdsXf_00140].

**[CP_SWS_DdsXf_00139] DDS Transformer in a transformer chain**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function DdsXf_<transformerId> specified in [CP_SWS_DdsXf_00138] shall exist for the first reference in the list of ordered references transformerChain from a DataTransformation to a TransformationTechnology if the DataTransformation is referenced by an ISignal in the role dataTransformation where the ISignal references a SystemSignal which is referenced by SenderReceiverToSignalMapping.⌋

**Note**. [CP_SWS_DdsXf_00139] It represents a restriction of the more general [SWS_Xfrm_00037] in [4, TransformerGeneral].

**[CP_SWS_DdsXf_00152] DDS Transformer in a DataPrototypeMapping**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function DdsXf_<transformerId> specified in [CP_SWS_DdsXf_00138] shall exist according to [SWS_Xfrm_00106] in [4, TransformerGeneral].⌋

**[CP_SWS_DdsXf_00140] DDS Transformer data marshalling procedure**

*Upstream requirements:* SRS_Xfrm_00301

⌈The function DdsXf_<transformerId> specified in [CP_SWS_DdsXf_00138] shall perform a raw copy of the primitive or complex data elements of SenderReceiverInterface communication into the provided buffer according to [CP_SWS_DdsXf_00728], [CP_SWS_DdsXf_00730], [CP_SWS_DdsXf_00731], [CP_SWS_DdsXf_00732], [CP_SWS_DdsXf_CONSTR_00733], [CP_SWS_DdsXf_CONSTR_00734]⌋

#### 8.3.1.2 DdsXf_Inv_<transformerId>

**[CP_SWS_DdsXf_00144] Definition of API function DdsXf_Inv_<transformerId>**

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

⌈

| | | |
|---|---|---|
| ***Service Name*** | DdsXf_Inv_<transformerId> | |
| ***Syntax*** | `uint8 DdsXf_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 marshalled data are stored by the Rte |
| | bufferLength | Used length of the buffer |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | dataElement | Data element which is the result of the transformation and contains the unmarshalled data element |
| ***Return value*** | uint8 | 0x00 (E_OK): Unmarshalling successful 0x01 (E_NO_DATA): No data available which can be unmarshalled 0x89 (E_SER_MALFORMED_MESSAGE): The received message is malformed. The transformer is not able to produce an output. |
| ***Description*** | This function unmarshalls a Sender/Receiver communication using the unmarshalling procedure of Dds. It takes the uint8 array containing the marshalled data as input and outputs the original data element which will be passed to the RTE. | |
| ***Available via*** | DdsXf.h | |

⌋

**[CP_SWS_DdsXf_00231] DdsXf_Inv_<transformerId> parameters definition**

*Upstream requirements:* SRS_Xfrm_00002

⌈In function `DdsXf_<transformerId>` defined in [CP_SWS_DdsXf_00138] the parameters:

- `type`
- `transformerId`

shall be defined according to [SWS_Xfrm_00042] in [4, TransformerGeneral].⌋

This function specified in [CP_SWS_DdsXf_00144] exists for each transformed `SenderReceiverInterface` communication which uses the Dds unmarshalling of `VariableDataPrototype`s procedure according to [CP_SWS_DdsXf_00147].

**[CP_SWS_DdsXf_00146] DDS inverse Transformer in a chain**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00144] shall exist for the first reference in the list of ordered references `transformerChain` from a `DataTransformation` to a `TransformationTechnology` if the `DataTransformation` is referenced by an `ISignal` in the role `dataTransformation` where the `ISignal` references a `SystemSignal` which is referenced by `SenderReceiverToSignalMapping`.⌋

**Note**. [CP_SWS_DdsXf_00146] represents a restriction of the more general [SWS_Xfrm_00043] in [4, TransformerGeneral].

**[CP_SWS_DdsXf_00153] DDS inverse Transformer in a DataPrototypeMapping**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00138] shall exist according to [SWS_Xfrm_00107] in [4, TransformerGeneral].⌋

**[CP_SWS_DdsXf_00147] DDS Transformer data unmarshalling procedure**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00144] shall retrieve the set of primitive or complex data elements of `SenderReceiverInterface` from a provided buffer created according to [CP_SWS_DdsXf_00140]⌋

**[CP_SWS_DdsXf_00264] DDS Transformer data unmarshalling with empty buffer**

*Upstream requirements:* SRS_Xfrm_00001, SRS_Xfrm_00004

⌈If `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00144] is called with buffer equal to `NULL_PTR` and bufferLength equal to 0, the output buffer shall not be changed and `DdsXf_Inv_<transformerId>` shall return with `E_NO_DATA`.⌋

### 8.3.2 ClientServerInterface API

#### 8.3.2.1 DdsXf_<transformerId>

**[CP_SWS_DdsXf_00141] Definition of API function DdsXf_<transformerId>**

*Upstream requirements:* SRS_Xfrm_00301

⌈

| | | |
|---|---|---|
| ***Service Name*** | DdsXf_<transformerId> | |
| ***Syntax*** | `uint8 DdsXf_<transformerId> (`<br>`  const Rte_Cs_TransactionHandleType* TransactionHandle,`<br>`  uint8* buffer,`<br>`  uint32* bufferLength,`<br>`  Std_ReturnType returnValue,`<br>`  <paramtype> data_1,`<br>`  <paramtype> data_n`<br>`)` | |
| ***Service ID [hex]*** | 0x08 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransactionHandle | Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests. |
| | returnValue | Return value from server side for transmission to the calling client. This argument is only available for marshallers of the response of a Client/Server communication if<br>• the ClientServerOperation has at least one PossibleError defined or<br>• autonomous error reaction is activated |
| | data_1 | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| | data_n | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | buffer | Buffer allocated by the RTE, where the transformed data has to be stored by the transformer |
| | bufferLength | Used length of the buffer |
| ***Return value*** | uint8 | 0x00 (E_OK): Marshalling successful |
| ***Description*** | This function transforms a Client/Server communication using the marshalling procedure of Dds. It takes the operation arguments and optionally the return value as input and outputs a uint8 array containing the marshalled data. The length of the marshalled 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*** | DdsXf.h | |

⌋

**[CP_SWS_DdsXf_00229] DdsXf_<transformerId> parameters definition**

*Upstream requirements:* SRS_Xfrm_00301

⌈In function `DdsXf_<transformerId>` defined in [CP_SWS_DdsXf_00141] the parameters:

- `paramtype`
- `type`

- `transformerId`

shall be defined according to [SWS_Xfrm_00038] in [4, TransformerGeneral].⌋

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.

This function specified in [CP_SWS_DdsXf_00141] exists for the server and each client of each transformed `ClientServerInterface` communication which uses the Dds marshalling of `VariableDataPrototype`s, `ClientServerOperation`s procedure according to [CP_SWS_DdsXf_00143]

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

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

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

**[CP_SWS_DdsXf_00142] DDS Transformer in a transformer chain**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall be defined according to [SWS_Xfrm_00039] in [4, TransformerGeneral].⌋

Due to [CP_SWS_DdsXf_00142], the API of [CP_SWS_DdsXf_00141] exists both on client and server.

**[CP_SWS_DdsXf_00143] DDS Transformer operation parameters marshalling procedure**

*Upstream requirements:* SRS_Xfrm_00301

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall perform a raw copy of the primitive or complex operation arguments of `ClientServerInterface` communication into the provided buffer according to [CP_SWS_DdsXf_00728], [CP_SWS_DdsXf_00730], [CP_SWS_DdsXf_00731], [CP_SWS_DdsXf_00732], [CP_SWS_DdsXf_CONSTR_00733]. [CP_SWS_DdsXf_CONSTR_00734]⌋

**[CP_SWS_DdsXf_00169] DDS Transformer infrastructure header set**

*Upstream requirements:* SRS_Xfrm_00305

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall set the infrastructure header according to the format defined in [CP_SWS_DdsXf_00751]⌋

**[CP_SWS_DdsXf_00160] Infrastructure header returnValue set**

*Upstream requirements:* SRS_Xfrm_00301, SRS_Xfrm_00305

⌈If executed on `server` side, the function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall copy the `returnValue` input parameter into the `ReturnValue` field of the provided buffer according to the format defined in [CP_SWS_DdsXf_00751]⌋

**[CP_SWS_DdsXf_00163] Infrastructure header clientId set**

*Upstream requirements:* SRS_Xfrm_00301

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall copy the `clientId` field of the `TransactionHandle` data structure input parameter into the `requestId.clientId` field of the provided buffer according to the format defined in [CP_SWS_DdsXf_00751]⌋

**[CP_SWS_DdsXf_00164] Infrastructure header sequenceCounter set**

*Upstream requirements:* SRS_Xfrm_00301

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall copy the `sequenceCounter` field of the `TransactionHandle` data structure input parameter into the `requestId.sequenceCounter` field of the provided buffer according to the format defined in [CP_SWS_DdsXf_00751]⌋

**[CP_SWS_DdsXf_00168] Infrastructure header reserved set**

*Upstream requirements:* SRS_Xfrm_00301

⌈The function `DdsXf_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall set the `reserved` field of the provided buffer according to the format defined in [CP_SWS_DdsXf_00751] to `0`.⌋

#### 8.3.2.2 DdsXf_Inv_<transformerId>

**[CP_SWS_DdsXf_00145] Definition of API function DdsXf_Inv_<transformerId>**

*Upstream requirements:* SRS_Xfrm_00002

⌈

| ***Service Name*** | DdsXf_Inv_<transformerId> |
|---|---|
| ***Syntax*** | `uint8 DdsXf_Inv_<transformerId> (`<br>`  Rte_Cs_TransactionHandleType* TransactionHandle,`<br>`  const uint8* buffer,`<br>`  uint32* bufferLength,`<br>`  Std_ReturnType returnValue,`<br>`  <type>* data_1,`<br>`  <type>* data_n`<br>`)` |
| ***Service ID [hex]*** | 0x07 |

▽

△

| | | |
|---|---|---|
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | buffer | Buffer allocated by the RTE, where the still marshalled data are stored by the Rte |
| | bufferLength | Used length of the buffer |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | TransactionHandle | Transaction handle according to [SWS_Rte_08732] (clientId and sequenceCounter) needed to differentiate between multiple requests. |
| | returnValue | Return value from server side for transmission to the calling client. This argument is only available for marshallers of the response of a Client/Server communication if<br>• the ClientServerOperation has at least one PossibleError defined or<br>• autonomous error reaction is activated |
| | data_1 | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| | data_n | Client/Server operation argument which shall be transformed (in the same order as in the corresponding interface) |
| ***Return value*** | uint8 | 0x00 (E_OK): Unmarshalling successful 0x01 (E_NO_DATA): No data available which can be marshalled 0x89 (E_SER_MALFORMED_MESSAGE): The received message is malformed. The transformer is not able to produce an output. |
| ***Description*** | This function unmarshalls a Client/Server communication using the unmarshalling procedure of Dds. It takes the uint8 array containing the marshalled data as input and outputs the return value of the server runnable and the operation arguments which have to be passed from the server to the client. | |
| ***Available via*** | DdsXf.h | |

⌋

**[CP_SWS_DdsXf_00232] DdsXf_Inv_<transformerId> parameters definition**

*Upstream requirements:* SRS_Xfrm_00002

⌈In function `DdsXf_Inv_<transformerId>` defined in [CP_SWS_DdsXf_00145] the parameters:

- `paramtype`
- `type`
- `transformerId`

shall be defined according to [SWS_Xfrm_00044] in [4, TransformerGeneral].⌋

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

This function specified in [CP_SWS_DdsXf_00141] exists for the server and each client of each transformed `ClientServerInterface` communication which uses the Dds unmarshalling of `VariableDataPrototype`s, `ClientServerOperation`s procedure according to [CP_SWS_DdsXf_00149]

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

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

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

**[CP_SWS_DdsXf_00148] DDS inverse Transformer in a chain**

*Upstream requirements:* SRS_Xfrm_00301

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] shall be defined according to [SWS_Xfrm_00045] in [4, TransformerGeneral].⌋

Due to [CP_SWS_DdsXf_00148], the API of [CP_SWS_DdsXf_00145] exists both on client and server.

**[CP_SWS_DdsXf_00149] DDS Transformer operation parameters unmarshalling procedure**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] shall retrieve the set of primitive or complex operation arguments in a `ClientServerInterface` from a provided buffer created according to [CP_SWS_DdsXf_00143], [CP_SWS_DdsXf_00169].⌋

**[CP_SWS_DdsXf_00170] DDS Transformer infrastructure header get**

*Upstream requirements:* SRS_Xfrm_00305

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00141] shall get the infrastructure header from a provided buffer created according to [CP_SWS_DdsXf_00143], [CP_SWS_DdsXf_00160]⌋

**[CP_SWS_DdsXf_00161] DDS Transformer returnValue get**

*Upstream requirements:* SRS_Xfrm_00002, SRS_Xfrm_00305

⌈If executed on client side, the function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] shall copy the `ReturnValue` from a provided buffer created according to [CP_SWS_DdsXf_00143], [CP_SWS_DdsXf_00169] into the output parameter `*returnValue`.⌋

**[CP_SWS_DdsXf_00165] DDS Transformer clientId get**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] shall copy the `requestId.clientId` from a provided buffer created according

to [CP_SWS_DdsXf_00143], [CP_SWS_DdsXf_00169] into the output parameter `TransactionHandle->clientId`.⌋

**[CP_SWS_DdsXf_00166] DDS Transformer sequenceCounter get**

*Upstream requirements:* SRS_Xfrm_00002

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] shall copy the `requestId.sequenceCounter` from a provided buffer created according to [CP_SWS_DdsXf_00143], [CP_SWS_DdsXf_00169] into the output parameter `TransactionHandle->sequenceCounter`.⌋

**[CP_SWS_DdsXf_00167] Infrastructure header reserved field check**

*Upstream requirements:* SRS_Xfrm_00305

⌈The function `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] shall return with `E_SER_MALFORMED_MESSAGE` and the output buffer shall not be changed if the provided buffer, created according to [CP_SWS_DdsXf_00143], [CP_SWS_DdsXf_00169], contains a `reserved` field not set to `0`.⌋

**[CP_SWS_DdsXf_00265] DDS Transformer operation parameters unmarshalling procedure with empty buffer**

*Upstream requirements:* SRS_Xfrm_00001, SRS_Xfrm_00004

⌈If `DdsXf_Inv_<transformerId>` specified in [CP_SWS_DdsXf_00145] is called with buffer equal to `NULL_PTR` and bufferLength equal to `0`, the output buffer shall not be changed and `DdsXf_Inv_<transformerId>` shall return with `E_NO_DATA`.⌋

## 8.4 Scheduled functions

DDS Transformer has no scheduled functions

## 8.5 Expected interfaces

There are no expected interfaces.

# 9 Sequence diagrams

There are no sequence diagrams.

# 10 Configuration specification

There is no module specific configuration available to the Dds. The EcuC defined in [4, SWS Transformer General] shall be used.

## 10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in [10].

## 10.2 Containers and configuration parameters

This specification does not have any configuration parameters.

## 10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in [10].

# A Not applicable requirements

**[SWS_DdsXf_NA_00000] Not applicable requirements**

*Upstream requirements:* SRS_BSW_00004, SRS_BSW_00101, SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00171, SRS_BSW_00323, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00344, SRS_BSW_00345, SRS_BSW_00375, SRS_BSW_00380, SRS_BSW_00384, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00409, SRS_BSW_00416, SRS_BSW_00419, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00450, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00458, SRS_BSW_00461, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00478, SRS_BSW_00488, SRS_BSW_00489, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00493, SRS_BSW_00496

⌈This item lists all the not applicable requirements for this specification.⌋

# B Change history of AUTOSAR traceable items

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

## B.1 Change History R25-11

### B.1.1 Added Specification Items in R25-11

[CP_SWS_DdsXf_00138] [CP_SWS_DdsXf_00139] [CP_SWS_DdsXf_00140] [CP_SWS_DdsXf_00141] [CP_SWS_DdsXf_00142] [CP_SWS_DdsXf_00143] [CP_SWS_DdsXf_00144] [CP_SWS_DdsXf_00145] [CP_SWS_DdsXf_00146] [CP_SWS_DdsXf_00147] [CP_SWS_DdsXf_00148] [CP_SWS_DdsXf_00149] [CP_SWS_DdsXf_00152] [CP_SWS_DdsXf_00153] [CP_SWS_DdsXf_00160] [CP_SWS_DdsXf_00161] [CP_SWS_DdsXf_00163] [CP_SWS_DdsXf_00164] [CP_SWS_DdsXf_00165] [CP_SWS_DdsXf_00166] [CP_SWS_DdsXf_00167] [CP_SWS_DdsXf_00168] [CP_SWS_DdsXf_00169] [CP_SWS_DdsXf_00170] [CP_SWS_DdsXf_00180] [CP_SWS_DdsXf_00183] [CP_SWS_DdsXf_00184] [CP_SWS_DdsXf_00228] [CP_SWS_DdsXf_00229] [CP_SWS_DdsXf_00231] [CP_SWS_DdsXf_00232] [CP_SWS_DdsXf_00264] [CP_SWS_DdsXf_00265] [CP_SWS_DdsXf_00266] [CP_SWS_DdsXf_00728] [CP_SWS_DdsXf_00730] [CP_SWS_DdsXf_00731] [CP_SWS_DdsXf_00732] [CP_SWS_DdsXf_00738] [CP_SWS_DdsXf_00740] [CP_SWS_DdsXf_00741] [CP_SWS_DdsXf_00742] [CP_SWS_DdsXf_00751] [CP_SWS_DdsXf_91002]

### B.1.2 Changed Specification Items in R25-11

none

### B.1.3 Deleted Specification Items in R25-11

none

### B.1.4 Added Constraints in R25-11

[CP_SWS_DdsXf_CONSTR_00733] [CP_SWS_DdsXf_CONSTR_00734] [CP_SWS_DdsXf_CONSTR_00735] [SWS_DdsXf_CONSTR_00151]

### B.1.5 Changed Constraints in R25-11

none

### B.1.6 Deleted Constraints in R25-11

none

## B.2 Change History R26-11

### B.2.1 Added Specification Items in R26-11

none

### B.2.2 Changed Specification Items in R26-11

none

### B.2.3 Deleted Specification Items in R26-11

[CP_SWS_DdsXf_00180]

### B.2.4 Added Constraints in R26-11

none

### B.2.5 Changed Constraints in R26-11

none

### B.2.6 Deleted Constraints in R26-11

none

# C Referenced Meta Classes

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

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

**Table C.1: ClientServerInterface**

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

▽

△

| *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.<br>This situation can happen during sliced execution or while output parameters are arrays (call by reference). The value true means that the `ClientServerOperation` is aware of the usage of a shared buffer and takes precautions to avoid unintentional overwrite of input arguments.<br>If the attribute does not exist or is set to false the `ClientServerOperation` does not have to consider the usage of a shared buffer.<br>This Attribute is only used by the AUTOSAR Classic Platform. |
| possibleError | ApplicationError | * | ref | Possible errors that may by raised by the referring operation.<br>**Tags:** atp.Splitkey=possibleError<br>This Attribute is only used by the AUTOSAR Classic Platform. |

**Table C.2: ClientServerOperation**

| *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.<br>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`.<br>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.<br>**Tags:** atp.Splitkey=firstDataPrototype |

▽

△

| *Class* | **DataPrototypeMapping** | | | |
|---|---|---|---|---|
| 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.<br>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.<br>**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.<br>**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.<br>**Tags:** atp.Splitkey=secondToFirstDataTransformation |
| subElement Mapping | SubElementMapping | * | aggr | This represents the owned SubelementMapping.<br>**Tags:** atp.Splitkey=subElementMapping |
| textTable Mapping | TextTableMapping | * | aggr | Applied TextTableMapping(s)<br>**Tags:** atp.Splitkey=textTableMapping |

**Table C.3: 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 | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| data Transformation Kind | DataTransformationKind Enum | 0..1 | attr | This attribute controls the kind of DataTransformation to be applied. |
| executeDespite Data Unavailability | Boolean | 0..1 | attr | Specifies whether the transformer chain is executed even if no input data are available. |
| transformer Chain (ordered) | Transformation Technology | * | ref | This attribute represents the definition of a chain of transformers that are supposed to be executed according to the order of being referenced from DataTransformation. |

**Table C.4: DataTransformation**

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

△

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

**Table C.5: ISignal**

| *Class* | **ImplementationDataType** | | | |
|---|---|---|---|---|
| ***Note*** | Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code.<br>**Tags:** atp.recommendedPackage=ImplementationDataTypes | | | |
| ***Base*** | *ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *Autosar DataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| dynamicArray SizeProfile | String | 0..1 | attr | Specifies the profile which the array will follow in case this data type is a variable size array. |

▽

△

| *Class* | **ImplementationDataType** | | | |
|---|---|---|---|---|
| isStructWith Optional Element | Boolean | 0..1 | attr | This attribute is only valid if the attribute category is set to STRUCTURE.<br>If set to true, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional. |
| subElement (ordered) | ImplementationData TypeElement | * | aggr | Specifies an element of an array, struct, or union data type.<br>The aggregation of `ImplementationDataTypeElement` is subject to variability with the purpose to support the conditional existence of elements inside a `ImplementationDataType` representing a structure.<br>**Stereotypes:** atpVariation<br>**Tags:** vh.latestBindingTime=preCompileTime |
| symbolProps | SymbolProps | 0..1 | aggr | This represents the SymbolProps for the Implementation DataType.<br>**Tags:** atp.Splitkey=symbolProps, symbolProps.short Name |
| typeEmitter | NameToken | 0..1 | attr | This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions. |

**Table C.6: ImplementationDataType**

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

**Table C.7: 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*** |

▽

△

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

**Table C.8: SenderReceiverToSignalMapping**

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

**Table C.9: SystemSignal**

| *Class* | **TransformationTechnology** | | | |
|---|---|---|---|---|
| *Note* | A TransformationTechnology is a transformer inside a transformer chain.<br>**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.<br>**Tags:** atp.Splitkey=bufferProperties |

▽

△

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

**Table C.10: TransformationTechnology**

| *Class* | **TriggerInterface** | | | |
|---|---|---|---|---|
| ***Note*** | A trigger interface declares a number of triggers that can be sent by an trigger source.<br>**Tags:** atp.recommendedPackage=PortInterfaces | | | |
| ***Base*** | *ARElement*, *ARObject*, *AtpBlueprint*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable* | | | |
| ***Aggregated by*** | ARPackage.element | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| trigger | Trigger | * | aggr | The Trigger of this trigger interface.<br>**Tags:** atp.Splitkey=trigger, trigger.shortName |

**Table C.11: TriggerInterface**

| *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*, *MultilanguageReferrable*, *Referrable* | | | |
| ***Aggregated by*** | ApplicationInterface.indication, *AtpClassifier*.atpFeature, BswInternalBehavior.arTypedPerInstanceMemory, BswModuleDescription.providedData, BswModuleDescription.requiredData, BulkNvDataDescriptor.bulkNvBlock, DiagnosticSovdAccessArgument.contentObject, *InternalBehavior*.staticMemory, NvBlockDescriptor.ramBlock, NvDataInterface.nvData, SenderReceiverInterface.dataElement, ServiceInterface.event, SwcInternalBehavior.arTypedPerInstanceMemory, SwcInternalBehavior.explicitInterRunnableVariable, SwcInternalBehavior.implicitInterRunnableVariable | | | |
| ***Attribute*** | ***Type*** | ***Mult.*** | ***Kind*** | ***Note*** |
| initValue | ValueSpecification | 0..1 | aggr | Specifies initial value(s) of the VariableDataPrototype<br>**Tags:** atp.Splitkey=initValue |

**Table C.12: VariableDataPrototype**