Elektrobit

# EB tresos® AutoCore Generic 8

# Transformers documentation

product release 8.8.7

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

## Technical support

https://www.elektrobit.com/support

## Legal disclaimer

Confidential information.

# Table of Contents

# 1. Overview of EB tresos AutoCore Generic 8 Transformers documentation

Welcome to the EB tresos AutoCore Generic 8 Transformers (ACG8 Transformers) product documentation.

This document provides:

- Chapter 2, “Supported features“: list of features supported by the ACG8 Transformers
- Chapter 3, “ACG8 Transformers release notes“: release notes for the ACG8 Transformers modules
- Chapter 4, “ACG8 Transformers user guide“: background information and instructions
- Chapter 5, “ACG8 Transformers module references“: information about configuration parameters and the application programming interface

# 2. Supported features

## 2.1. Overview

This chapter provides an overview of the products of ACG8 Transformers and the features that are currently supported.

Section 2.2, “Product details” contains an overview of the products of ACG8 Transformers.

Section 2.3.1, “Supported ComXf features” contains an overview of `ComXf` features.

Section 2.3.2, “Supported SomeIpXf features” contains an overview of `SomeIpXf` features.

## 2.2. Product details

ACG8 Transformers provides AUTOSAR modules for the EB tresos AutoCore Generic (ACG) product line. ACG8 Transformers is based on AUTOSAR 4.2.1, selected features of AUTOSAR 4.2.2, AUTOSAR 4.3.0, and EB-specific enhancements implemented compatible to the AUTOSAR standard.

ACG8 Transformers includes the following basic software modules:

| **Basic software modules** | **Module abbreviation** |
|---|---|
| COM Based Transformer | ComXf |
| SOME/IP Transformer | SomeIpXf |

## 2.3. Feature details

This chapter contains an overview of the supported and unsupported features.

### 2.3.1. Supported ComXf features

`ComXf` provides the following main features according to the AUTOSAR specification:

- **Data serialization according to Com configuration:** Serialization of complex data elements into a byte stream of variable length according to the configuration of the AUTOSAR `Com` module which in turn is derived from the AUTOSAR system description.

- **Data de-serialization according to Com configuration:** De-serialization of a byte stream of variable length into complex data element according to the configuration of the AUTOSAR `Com` module which in turn is derived from the AUTOSAR system description.

### 2.3.2. Supported SomeIpXf features

`SomeIpXf` provides the following main features according to the AUTOSAR specification:

- **Data serialization according to SOME/IP:** Serialization of complex data elements and remote client/server calls and responses into a byte stream of variable length according to the SOME/IP specification. Hereby the serialization of remote client/server calls includes remote client/server calls of fire-and-forget methods (`REQUEST_NO_RETURN`) which are handled via inter-ECU external trigger event communication according to AUTOSAR 4.2.2 (RFC #67799).
- **Data de-serialization according to SOME/IP:** De-serialization of a byte stream of variable length into complex data elements and remote client/server calls and responses according to the SOME/IP specification. Hereby the de-serialization of remote client/server calls includes remote client/server calls of fire-and-forget methods (`REQUEST_NO_RETURN`) which are handled via inter-ECU external trigger event communication according to AUTOSAR 4.2.2 (RFC #67799).

# 3. ACG8 Transformers release notes

## 3.1. Overview

This chapter provides the ACG8 Transformers product specific release notes. General release notes that are applicable to all products are provided in the EB tresos AutoCore Generic documentation. Refer to the general release notes in addition to the product release notes documented here.

## 3.2. Scope of the release

### 3.2.1. Configuration tool

Your release of EB tresos AutoCore is compatible with the release of the EB tresos Studio configuration tool:

- EB tresos Studio: 29.2.0 b220916-0321

### 3.2.2. AUTOSAR modules

The following table lists the AUTOSAR modules that are part of this ACG8 Transformers release.

| Module name | AUTOSAR version and revision | SWS version and revision | Module version | Supplier |
|---|---|---|---|---|
| ComXf | 4.2.1 [] | 4.2.1 [0000] | 1.0.41 | Elektrobit Automotive GmbH |
| SomeIpXf | 4.2.1 [] | 4.2.1 [0000] | 1.0.55 | Elektrobit Automotive GmbH |

Table 3.1. Hardware-Independent Modules specified by the AUTOSAR standard

### 3.2.3. EB (Elektrobit) modules

The following table lists all modules which are part of this release but are not specified by the AUTOSAR standard. These modules include tooling developed by EB or they may hold files shared by all other modules.

| Module name | Module version | Supplier |
|---|---|---|
| Xfrm | 1.0.39 | Elektrobit Automotive GmbH |

Table 3.2. Modules not specified by the AUTOSAR standard

### 3.2.4. MCAL modules and EB tresos AutoCore OS

For information about MCAL modules and OS, refer to the respective documentation, which is available as PDF at `$TRESOS_BASE/doc/3.0_EB_tresos_AutoCore_OS` and `$TRESOS_BASE/doc/5.0_MCAL_modules`[1]. It is also available in the online help in EB tresos Studio. Browse to the folders `EB tresos AutoCore OS` and `MCAL modules`.

## 3.3. Module release notes

### 3.3.1. ComXf module release notes

- AUTOSAR R4.2 Rev 1
- AUTOSAR SWS document version: 4.2.1
- Module version: 1.0.41.B567464
- Supplier: Elektrobit Automotive GmbH

#### 3.3.1.1. Change log

This chapter lists the changes between different versions.

**Module version 1.0.41**
2022-10-12

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.40**
2022-01-28

[1] `$TRESOS_BASE` is the location at which you installed EB tresos Studio.

- ASCCOMXF-639, ASCXFRM-439 Fixed known issue: Code generation for multiple receivers results in an error

**Module version 1.0.39**
2021-11-12

- ASCCOMXF-628, ASCXFRM-434 Fixed known issue: Safety Com transformer forwards undersized buffer length to the E2EXf (Note: requires also Xfrm library update)

**Module version 1.0.38**
2021-10-08

- Added support of different SenderReceiverToSignalGroupMappings at outermost RPorts based on configured values in container XfrmVariableDataPrototypeInstanceRef

**Module version 1.0.37**
2021-03-05

- Added generation of unsigned suffix to the generated C data element path for array members
- Updated preprocessor include guards to be PC-lint compatible

**Module version 1.0.36**
2021-02-12

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.35**
2020-09-25

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.34**
2020-07-31

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.33**
2020-05-22

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.32**
2020-01-24

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.31**
2019-10-11

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.30**
2019-07-05

- ASCCOMXF-492, ASCE2E-771 Fixed known issue: Invalid safety-related ComXf support for XfrmBuffer-LengthType configured to UINT32 (Note: requires also E2E library update)

**Module version 1.0.29**
2019-03-22

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.28**
2019-02-15

- Added 64 bit data types de-/serialization support

**Module version 1.0.27**
2018-10-26

- ASCCOMXF-449 Fixed known issue: Generator aborts with an exception when simple array of primitive data types is configured as ApplicationDataTypes

**Module version 1.0.26**
2018-09-28

- ASCCOMXF-442 Fixed known issue: Safety Com transformer calculates wrong buffer size

**Module version 1.0.25**
2018-07-27

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.24**
2018-06-22

- Moved safety checker part from module to asc_ComXfCV

**Module version 1.0.23**
2018-05-25

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.22**
2018-03-16

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.21**
2018-02-16

- Allow partial SenderReceiverToSignalGroupMapping

**Module version 1.0.20**
2017-12-15

- ASCCOMXF-347 Fixed known issue: Wrong byte size used for packing/unpacking of data elements configured with big-endian byte order
- Implemented support for configurable type of BufferLength
- Added XfrmIsSafetyTransformer configuration parameter

**Module version 1.0.19**
2017-09-22

- Added 64 bit support for safety related de-/serialization of signal groups
- Switch from MISRA-C:2004 to MISRA-C:2012

**Module version 1.0.18**
2017-08-25

- Use ImplementationDataType of (outermost) composition

**Module version 1.0.17**
2017-06-30

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.16**
2017-06-02

- Added support for safety related de-/serialization of signal groups

**Module version 1.0.15**
2017-05-05

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.14**
2017-03-31

- Incorporated Bugzilla RfC 69896: Execution of Transformer chain in case of unqueued communication when no data is available
- Incorporated Bugzilla RfC 68623: Insufficient specification of autonomous error response
- Removed and incorporated EB tresos AutoCore Generic 8 Transformer (COM) user's guide into EB tresos AutoCore Generic 8 Transformers documentation

**Module version 1.0.13**
2017-01-05

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.12**
2016-12-02

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.11**
2016-11-04

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.10**
2016-10-07

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.9**
2016-09-09

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.8**
2016-07-01

- Added support for memory partitioning of partitioned RTE
- ASCCOMXF-115 Fixed known issue: Incorrect mapping from ApplicationRecordElement to ImplementationRecordElement

**Module version 1.0.7**
2016-05-25

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.6**
2016-04-29

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.5**
2016-04-01

- ASCCOMXF-64 Fixed known issue: Unserialized signals when referenced by application data type

**Module version 1.0.4**
2016-02-05

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.3**
2016-01-15

- Added support of ApplicationDataTypes

**Module version 1.0.2**
2015-11-06

- Changed parameter name XfrmTransformationBswModuleEntryRef to XfrmTransformerBswModuleEntryRef (see AUTOSAR RfC #68531)
- Added support for specification of XfrmVariableDataPrototypeInstanceRef (multiple receivers)

**Module version 1.0.1**
2015-10-09

- Implemented robust code generation by reporting a warning and ignoring problematic XfrmImplementationMappings

**Module version 1.0.0**
2015-06-19

- Initial version

### 3.3.1.2. New features

- No new features have been added since the last release.

#### 3.3.1.3. Elektrobit-specific enhancements

This chapter lists the enhancements provided by the module.

- **Integration requirement: EB_INTREQ_ComXf_0002**

  Support for serialization of unaligned / non-consecutively mapped signal groups

  Description:

  The ACG7 Transformer (COM) supports the serialization of signal groups which do not start or end at byte boundaries. Further, there is no constraint that group signals must be completely or consecutively mapped to the IPdu.

  Before serialization the transformer initializes the (Rte) buffer with the latest values of the area of the EB Com module internal I-PDU buffer corresponding to the signal group. That includes unused area bits as well as the values of signals / group signals of other group signals. If the value of a signal / group signal has never been updated the latest value equals the initial value (ComSignalInitValue). Therefore the initial value is also taken for group signals which are not mapped to a Data Element member.

  Note: Unaligned signal groups must be considered in the AUTOSAR System / Software Component Description such that the TransformerTechnology of the related COM Based Transformer specifies more bytes for the BufferComputation within the BufferProperties.

  Note: There is no check (neither at configuration time, nor at compile time, nor at runtime) if the buffer provided by Rte is big enough to hold the resulting array starting with the byte of the first group signal and ending with the last group signal.

- **Integration requirement: EB_INTREQ_ComXf_0003**

  Support for de-serialization of unaligned / non-consecutively mapped signal groups

  Description:

  The ACG7 Transformer (COM) supports the de-serialization of signal groups which do not start or end at byte boundaries. Further, there is no constraint that group signals must be completely or consecutively mapped to the IPdu.

  The AUTOSAR Com module updates the buffer for the transformer with the latest values of the area of the AUTOSAR Com module internal I-PDU buffer corresponding to the signal group. That includes unused area bits as well as the values of signals / group signals of other group signals. The ACG7 Transformer (COM) extracts only the group signals which are mapped to a data element.

  Note: Unaligned signal groups must be considered in the AUTOSAR System / Software Component Description such that the TransformerTechnology of the related COM Based Transformer specifies more bytes for the BufferComputation within the BufferProperties.

- **Integration requirement: EB_INTREQ_ComXf_0004**

Support for safety related de-/serialization of signal groups

Description:

If a basic software entry of the ComXf module references a transformer which is part of a transformer chain that includes the E2EXf transformer (safety related), the basic software entry of the ComXf module uses the pack and unpack macros from the E2E library for safety related serialization and de-serialization of the composite data type.

In order to provide safety related serialization and de-serialization the ComXf module does not use the post-build configuration or library APIs of the EB Com module. The system configuration and the pack and unpack macros from the E2E library are used instead. This ensures no information dependencies between the ComXf module and the Com module in case of safety related serialization and de-serialization. However, the configuration of signal groups within the system configuration (for the ComXf module) and the Com module have to be consistent which implies that changes done within the local Com configuration regarding signal groups (i.e. ComBitposition or data types of group signals) needs also be adapted within the system configuration (i.e. ISignalToIPduMapping).

The E2E pack macros including range checks for each group signal during serialization of the corresponding data element member. The range check occur if the value of the data element member cannot be represented by the mapped group signal in relation to the configured bit-length and signal type. In this case the ComXf module returns the error code E2E_RANGECHK_INVALID (0xFFU).

Unused area bits are updated with the value specified in parameter UNUSED-BIT-PATTERN of the system configuration.

- Configurable data type for BufferLength

Description:

Module configuration parameter `XfrmBufferLengthType` enables the user to adjust the data type for parameter `BufferLength` of the transformer APIs.

Configuring `XfrmBufferLengthType` to `UINT16` sets the data type of parameter `BufferLength` to `uint16`.

Configuring `XfrmBufferLengthType` to `UINT32` sets the data type of parameter `BufferLength` to `uint32`.

Rationale:

This module shall be able to serialize and deserialize data with a size greater than 64 KiB.

- Configurable safety relevance of transformer

Description:

Module configuration parameter `XfrmIsSafetyTransformer` enables the user to explicitly state if the transformer is classified as safety relevant (ASILs) or not (QM).

If parameter `XfrmIsSafetyTransformer` is enabled and configured to `TRUE`, the transformer is considered as a transformer classified for handling data with safety relevance (ASILs).

If parameter `XfrmIsSafetyTransformer` is enabled and configured to `FALSE`, the transformer is considered as a transformer classified for handling data without safety relevance (QM).

If parameter `XfrmIsSafetyTransformer` is disabled, the transformer is only considered as a transformer classified for handling data with safety relevance (ASILs), when a safety transformer with parameter `disableEndToEndCheck` set to `FALSE` is present within the same `DataTransformation`. Otherwise the transformer is considered as a transformer classified for handling data without safety relevance (QM).

Rationale:

This enhancement was introduced in order to provide a clear configuration semantics for the user.

- Memory partitioning

  Description:

  Module configuration parameter `XfrmOsApplicationRef` enables the user to assign the transformer to a dedicated memory partition.

  If parameter `XfrmOsApplicationRef` is enabled and a valid reference to an `OsApplication` is provided, the global variables of the transformer are mapped to the memory partition to which the referenced `OsApplication` belongs.

  If parameter `XfrmOsApplicationRef` is disabled, the global variables of the transformer are mapped to the default memory partition.

  Rationale:

  This enhancement was introduced in order to support the memory partitioning of the RTE.

### 3.3.1.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- Module Interlink Types Header File

  Description:

  The module interlink types header file `SchM_ComXfType.h` is included instead of the header file `SchM_ComXf_Type.h` specified by SWS_ComXf_00001.

Rationale:

This allows compatibility to the Rte which generates header file `SchM_ComXfType.h`.

Requirements:

SWS_ComXf_00001

- Non-reentrant transformer APIs

Description:

In contrast to AUTOSAR which specifies that the generated serializer and de-serializer APIs are reentrant, these APIs are non-reentrant. Additional information on race conditions are stated in Integration requirement EB_INTREQ_Com_0002 of the AUTOSAR Com module.

- Extended Production Errors

Description:

Extended production errors are not supported.

Requirements:

ECUC_Xfrm_00016 ECUC_Xfrm_00015 SWS_Xfrm_00070 SWS_Xfrm_00071

- Error Handling

Description:

Even though a hard error is returned, the output buffer of the serializer transformer API (SWS_ComXf_00007) will be changed. This means that the serialized data is written to the output buffer during serialization. If the hard error occurs the buffer where the serialized data is written to, will not be used.

Rationale:

Only one buffer is used for the serialization process which leads to better performance.

Requirements:

SWS_Xfrm_00051

- **The following Com module deviations also apply for the ComXf module**:
- Signal invalidation is not supported (but is supported via RTE) (reference to product description: ASCPD-15)

Description:

Signal invalidation is not supported. However, the EB tresos AutoCore RTE is extended in order to provide the signal invalidation functionality based on the configuration of the `Com` module.

Requirements:

COM099, COM286, COM680, COM681, COM736, COM683, COM737, COM717, COM718, COM334, COM024, COM203, COM642, COM643, COM288, COM644, COM557, COM645, COM536, COM315_Conf, COM391_Conf, COM314_Conf COM738, COM682, COM483, COM396, COM005, COM731

- `Com_SendSignal()` does not return `COM_SERVICE_NOT_AVAILABLE` in case the value of the signal does not fit into the PDU

Description:

The function `Com_SendSignal()` does not return `COM_SERVICE_NOT_AVAILABLE` in case the value of the signal does not fit into the PDU, but an error is reported to DET. However, the SWS states: Return value: `E_OK` - service has been accepted `COM_SERVICE_ NOT_AVAILABLE` - corresponding I-PDU group was stopped (or service failed due to development error). Therefore a `COM_SERVICE_ NOT_AVAILABLE` should be returned.

Requirements:

COM197

- Restricted support of small Rx-I-PDUs

Description:

According to the AUTOSAR COM SWS chapter *Signal indication (Unpacking of I-PDUs)* it is specified that it is allowed that smaller than expected Rx-I-PDUs can be received (configured). In such a case partly or not received signals/signal groups shall not be updated and no notification via `ComNotification` shall take place.

However, the implementation behaves as follows:

- The received data length (`PduInfoPtr->SduLength`) is copied into the Com-internal I-PDU buffer. If a signal or signal groups are received only partly, these are also updated partly. Signals/signal groups which are not received at all are not updated. If the I-PDU contains a dynamic length signal the API Com_ReceiveDynSignal() does not copy data and 0 is returned in the length parameter. If a dynamic length signal is used within the signal gateway, the length of the corresponding Tx dynamic length signal is also set to 0.
- `ComNotification` is invoked for all signals or signal groups that belong to this Rx-I-PDU.

Workaround 1:

- For a smaller Rx-I-PDU, for which it is expected that a signal or signal group is only partly updated: Configure an I-PDU callout which updates the partly received signal/signal groups with a proper value (either last received value or initial value).

- Design the applications in a way that they can handle `ComNotification` of signals which are not or only partly received.

Workaround 2:

- Provide an application for each expected size of the Rx-I-PDU.
- For each expected size of the Rx-I-PDU configure a Rx-I-PDU in the Com module.
- Create a mapping between the additional applications and Rx-I-PDUs.
- Configure an Rx-I-PDU callout with the large I-PDU which invocates `Com_RxIndication` of the respective smaller Rx-I-PDU and returns `FALSE` in case a shorter I-PDU is received.

Rationale:

In general this limitation allows a more efficient implementation for I-PDUs which are received completely. Workarounds are available if this feature is required.

Requirements:

COM574, COM575

- No generation of symbolic name value into `Com_Cfg.h`

Description:

Several requirements claim that the symbolic names for the Com Handle IDs shall be published via `Com_Cfg.h`. However, the symbolic name values are provided in `Com_SymbolicNames_PBcfg.h` which is also included in `Com.h`.

Rationale:

- Requirement is a deviation against TPS_ECUC_02108 of Specification of ECU Configuration which says that the symbolic name values shall be generated into the module header file.
- Requirement is a deviation against SWS_BSW_00200 of SWS General Specification of Basic Software Modules AUTOSAR 4.1 Rev 1, which says that symbolic name values shall be imported through the header of the BSW module that provides the value.
- Shall be removed in future AUTOSAR releases, see http://www.autosar.org/bugzilla/show_bug.cgi?id=60888

Requirements:

COM174, COM126, COM163, COM044, COM521

- No support of dynamic length signals in signal groups

Description:

Dynamic length signals are only supported as signals. They are not supported in a group signal.

Rationale:

The implementation uses `Com_UpdateShadowSignal()` and `Com_ReceiveShadowSignal()` for the access of group signals. Since AUTOSAR does not define an equivalent API for access dynamic group signals, it is not possible to support dynamic length signals for group signals.

Requirements:

COM127

- No support of zero size signals / group signals with transfer property PENDING

Description:

In contrast to AUTOSAR which allows zero size signals / group signals for transfer properties TRIGGERED, PENDING, and TRIGGERED_WITHOUT_REPETITION, only transfer property TRIGGERED is supported.

Requirements:

COM762

- Overlapping of ComSignals / ComGroupSignals

Description:

In contrast to AUTOSAR which states that ComSignal / ComGroupSignal are not allowed to overlap each other, the COM module allows the configuration of overlapped ComSignals / ComGroupSignals.

Requirements:

COM102

- Configurable callback / callout functions are not provided in Com_Cbk.h

Description:

In contrast to AUTOSAR which states that the configurable callback and callout functions shall be provide in the header file Com_Cbk.h, the COM module does not declare these functions. Instead, it simply declares and calls these external function only in an internal Com compilation unit.

Rationale:

These functions are usually generated / implemented by the Rte which also generates adequate function declarations. The linker then is able to resolve the function calls and the adequate function definitions in Rte.

Requirements:

COM731

- Violation of VSMD rule Constr_3023

Description:

The attribute apiServicePrefix is mandatory for VSMDs derived from the CDD StMD. The attribute shall not be provided for VSMDs derived from any other StMDs. The rule is based on Constr_3023 from AUTOSAR_TPS_ECUConfiguration.pdf of 4.2.1 Release.

Effected node:

- StMD-Node: `/AUTOSAR/EcucDefs/Xfrm`

Rationale:

The module violates the second part of the constraint, but correctly, because SWS_ComXf_00025 requires the definition of attribute apiServicePrefix. The discrepancy is handled by AUTOSAR with https://jira.autosar.org/browse/AR-109503 .

Requirements:

SWS_ComXf_00025

- Violation of VSMD rule EcucSws_2038_2040_ASR41

Description:

If there is a SYMBOLIC-NAME-REFERENCE which points to another module, the rule EcucSws_2038_2040_ASR41, which is based on requirements TPS_ECUC_02038 and TPS_ECUC_02040 from AUTOSAR_TPS_ECUConfiguration.pdf of 4.1 Release, always creates a violation.

Effected nodes:

- VSMD-Node: `Xfrm/XfrmImplementationMapping/XfrmDemEventParameterRefs/XFRM_E_MALFORMED_MESSAGE`
- VSMD-Node: `Xfrm/XfrmImplementationMapping/XfrmOsApplicationRef`

Rationale:

Violation occurs due to an invalid behavior within the EB tresos Studio. No useful workaround available. Rule EcucSws_2038_2040_ASR41 shall be ignored.

Requirements:

ECUC_Xfrm_00016

#### 3.3.1.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- Limitation on Implementation Mapping

  Description:

  The short name of the implementation mapping has to be identical with the short name of the referenced BSW module entry. I.e. the value of parameter XfrmImplementationMapping is the same as the referenced XfrmTransformerBswModuleEntryRef (or XfrmInvTransformerBswModuleEntryRef).

  As a consequence, short name of XfrmImplementationMapping has to be equal the function name of the related transformer API.

  Rationale:

  It eases the implementation of the module.

- Limitation on safety related de-/serialization of signal groups regarding configuration aspects

  Description:

  In order to provide correct safety related serialization and de-serialization the configuration of signal groups within the system configuration (for the ComXf module) and the Com module have to be consistent which implies that changes done within the local Com configuration regarding signal groups (i.e. ComBitposition or data types of group signals) needs also be adapted within the system configuration (i.e. ISignalToIPduMapping).

  Rationale:

  For safety related serialization and de-serialization the ComXf module does not use the post-build configuration or library APIs of the EB Com module. The system configuration and the pack and unpack macros from the E2E library are used instead. This ensures no information dependencies between the ComXf module and the Com module in case of safety related serialization and de-serialization.

- Limitation on safety related de-/serialization of signal groups regarding range checks

  Description:

  The E2E pack macro range checks for float and byte array (opaque data type UINT8_N) signal types are not supported.

  Range checks for 64 bit signal types are only supported on byte level.

- Limitation on safety related de-/serialization of signal groups regarding 64 bit support

  Description:

The E2E packing and unpacking macros for 64 bit signal types includes the following limitations:

- group signals shall be byte aligned
- bitlength shall be a multiple of 8

- Limitation on safety related de-/serialization of signal groups regarding development errors

  Description:

  In case of safety related serialization and de-serialization of signal groups, development errors are not supported. Development error checks are always performed, but development errors will not be reported. In case of an error a transformer and an inverted transformer return E_SER_GENERIC_ERROR. The configuration parameter XfrmDevErrorDetect will be ignored.

  Rationale:

  Development errors would have to be conform with the highest requested saftey standard.

- Limitation on safety related de-/serialization of signal groups regarding unaligned / non-consecutively mapped signal groups

  Description:

  In case of safety related serialization and de-serialization of unaligned / non-consecutively mapped signal groups, the transformer does not initialize the (Rte) buffer with the latest values of the area of the EB Com module internal I-PDU buffer corresponding to the signal group or the values of signals / group signals of other group signals.

  Rationale:

  The initialization of the buffer with the latest version needs internal EB Com module information. This shall not apply for safety related de-/serialization of signal groups. The bit areas are handled as unused areas.

- Implementation-specific restrictions

  Description:

  There are some implementation-specific restrictions which are listed for completeness only, as they are most probably irrelevant for the intended use of the module:

  - The maximum number of signals allowed is 65534.
  - The maximum number of Rx/Tx I-PDUs allowed is 65534.
  - The maximum number of callouts configured is 65534.
  - The sum of the lengths of all byte-arrays which are sent via the Com module must not exceed 65535 bytes.

- The number of signals and signal group members, signal groups, notifications per I-PDU must not exceed 254.

- Limitation on handling user data with more than 64 KiB

  Description:

  With parameter `XfrmBufferLengthType` configured to `UINT32`, the module is basically allowed to handle user data with more than 64 KiB.

  Setting the parameter `XfrmBufferLengthType` to `UINT32` is possible even though user data with more than 64 KiB shall not be handled.

  Rationale:

  The non-safety related serialization/deserialization relies on the COM module. This holds dedicated constraints on the signal length, not allowing user data greater than 64 KiB to be handled.

  The safety related serialization/deserialization relies on the EB tresos data base instead of the COM module. This approach, which would allow handling of user data greater than 64 KiB, is no use case in a safety environment.

- Restriction on 64 bit signals/group signals

  Description:

  The following restrictions for signals/group signals with `ComSignalType` configured to `UINT64` apply:

  - The `ComBitPosition` is restricted to be byte aligned.
  - The `ComBitSize` is restricted to be a multiple of 8 bit.
  - The `ComFilterAlgorithm` is limited to `ALWAYS`, `NEVER`, `ONE_EVERY_N`, `MASKED_NEW_DIFFERS_X` and `MASKED_NEW_EQUALS_X`.
  - For the `ComFilterAlgorithms` `MASKED_NEW_DIFFERS_X` and `MASKED_NEW_EQUALS_X`, only the bits with respect to the configured `ComBitSize` are taken into account for the filter evaluation.
  - The `ComFilterAlgorithm` for zero size signals / group signals is limited to `ALWAYS` and `NEVER`.

  The following restrictions for signals/group signals with `ComSignalType` configured to `SINT64` apply:

  - The `ComBitPosition` is restricted to be byte aligned.
  - The `ComBitSize` is restricted to be 64 bit (zero size signals / group signals are not supported).
  - The `ComFilterAlgorithm` is limited to `ALWAYS`, `NEVER`, `ONE_EVERY_N`, `MASKED_NEW_DIFFERS_X` and `MASKED_NEW_EQUALS_X`.

  Requirements:

COM675, COM602, COM170_Conf, COM352, COM325, COM764, COM273, COM603, COM302, COM303, COM763, COM222 COM324, COM793

#### 3.3.1.6. Open-source software

ComXf does not use open-source software.

## 3.3.2. SomeIpXf module release notes

- AUTOSAR R4.2 Rev 1
- AUTOSAR SWS document version: 4.2.1
- Module version: 1.0.55.B567464
- Supplier: Elektrobit Automotive GmbH

#### 3.3.2.1. Change log

This chapter lists the changes between different versions.

**Module version 1.0.55**
2022-10-12

- Internal module improvement. This module version update does not affect module functionality
- Removed return codes SOMEIPXF_E_UNKNOWN_SERVICE and SOMEIPXF_E_UNKNOWN_METHOD

**Module version 1.0.54**
2022-05-11

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.53**
2022-03-09

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.52**
2021-12-10

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.51**
2021-10-08

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.50**
2021-09-17

- ASCSOMEIPXF-1373 Fixed known issue: Deserialization aborts if the length field value of a serialized data element is greater than the received buffer length
- Implemented function to extract the relevant SOME/IP protocol header fields of the message and the type of the message result

**Module version 1.0.49**
2021-08-20

- ASCSOMEIPXF-1372 Fixed known issue: Serializer returns generic error for normal response without operation arguments

**Module version 1.0.48**
2021-07-28

- ASCSOMEIPXF-1361 Fixed known issue: Wrong Request ID in autonomous error response
- Optimized buffer length checks for basic data types

**Module version 1.0.47**
2021-06-25

- Improved code generators detection of invalid system model configuration related to Client/Server method arguments that use tag-length-value (TLV) encoding
- Implemented support for SOME/IP Message Error Callout interface

- Added new module configuration parameter to determine which value is inserted into the message type field in the header for an autonomous error response
- Implemented skipping of variable size array payload bytes based on the length field in case of variable size array length is greater than expected
- Added new module configuration parameter for enabling the check for duplicated Data IDs during deserialization
- Added new module configuration parameter for enabling the reordering of elements that use tag-length-value (TLV) encoding during deserialization
- Fixed memory stack size overhead for complex data types

**Module version 1.0.46**
2021-03-05

- Updated preprocessor include guards to be PC-lint compatible

**Module version 1.0.45**
2021-02-12

- Implemented major inspection source code findings
- Added support for nested variable size arrays as an indirect payload of another variable size array
- Added support for structures, extensible structures and linear variable size arrays as a payload of an extensible array
- Implemented support for tag-length-value (TLV) encoding
- Added new module configuration parameter for deriving the size of length fields of variable size arrays

**Module version 1.0.44**
2020-07-31

- Incorporated RfC 68113: Session ID is always set to 0x00 for S/R communication

**Module version 1.0.43**
2020-05-22

- ASCSOMEIPXF-1055 Fixed known issue: Serialization of fixed size arrays with array length fields results in a generic transformer error

**Module version 1.0.42**
2020-02-21

- ASCSOMEIPXF-283 Fixed known issue: Length field values get de-/serialized wrong on big endian platforms

**Module version 1.0.41**
2020-01-24

- ASCXFRM-365 Fixed known issue: Variable Size Arrays are serialized to the wrong array type

**Module version 1.0.40**
2019-12-06

- Implemented major inspection source code findings
- ASCSOMEIPXF-907 Fixed known issue: Wrong deserialization of extensible structures and arrays

**Module version 1.0.39**
2019-11-08

- ASCSOMEIPXF-964 Fixed known issue: External trigger event transformer leads to compile error within safe SomeIpXf partition

**Module version 1.0.38**
2019-10-11

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.37**
2019-05-17

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.36**
2019-03-22

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.35**
2019-02-15

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.34**
2018-10-26

- Added support for bitfields

**Module version 1.0.33**
2018-07-27

- Improved support for configurable type of BufferLength

**Module version 1.0.32**
2018-06-22

- Moved safety checker part from module to asc_SomeIpXfCV

**Module version 1.0.31**
2018-05-25

- Implemented non-functional code improvements to fix Misra violations
- ASCSOMEIPXF-822 Fixed known issue: Compile error in generated SomeIpXf_Api_gen.h

**Module version 1.0.30**
2018-03-16

- ASCSOMEIPXF-781 Fixed known issue: Client/server deserializer without arguments silently ignores hard errors
- Improved support for configurable type of BufferLength

**Module version 1.0.29**
2018-02-16

- ASCSOMEIPXF-765 Fixed known issue: Compile error for client/server sending transformer API with autonomous error response

**Module version 1.0.28**
2018-01-19

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.27**
2017-12-15

- Implemented support for configurable type of BufferLength
- Added XfrmIsSafetyTransformer configuration parameter
- Improved support for configurable type of BufferLength

**Module version 1.0.26**
2017-09-22

- Switch from MISRA-C:2004 to MISRA-C:2012
- Replaced DET handling with defensive programming mechanism returning E_SER_GENERIC_ERROR
- Modified initialization sequence
- Prepared SomeIpXf for use in safety releases
- ASCSOMEIPXF-660 Fixed known issue: Missing memory section for array holding the process sequence

**Module version 1.0.25**
2017-07-28

- Implemented usage of ApplicationDataType of (outermost) SwComponentType

**Module version 1.0.24**
2017-06-30

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.23**
2017-06-02

- ASCSOMEIPXF-600 Fixed known issue: No error message when length field exceeds buffer length of variable size array

**Module version 1.0.22**
2017-05-05

- ASCSOMEIPXF-560 Fixed known issue: Compilation error due to multiple definitions of identical configuration array size macros
- Added support for variable size array profile of type fully flexible
- ASCSOMEIPXF-561 Fixed known issue: Wrong ClientID and SessionID in client/server communication on big-endian platforms

**Module version 1.0.21**
2017-03-31

- Incorporated Bugzilla RfC 66764 in allowing usage of uint64 and sint64 data types.
- Incorporated Bugzilla RfC 68623: Insufficient specification of autonomous error response
- Incorporated Bugzilla RFC 69896: Execution of Transformer chain in case of unqueued communication when no data is available
- Incorporated Bugzilla RfC 72952: Adaptation of ApplicationErrors inconsistent
- Incorporated Bugzilla RfC 76926: Contradiction between SWS_SomeIpXf_00107 and constr_1108
- Removed and incorporated EB tresos AutoCore Generic 8 Transformer (SOME/IP) user's guide into EB tresos AutoCore Generic 8 Transformers documentation

**Module version 1.0.20**
2017-02-03

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.19**
2017-01-05

- Enhanced variable size array usage in allowing size indicator value 0

**Module version 1.0.18**
2016-12-02

- Implemented handling of missing data parameters in SOME/IP transformer functions

**Module version 1.0.17**
2016-11-04

- ASCSOMEIPXF-431 Fixed known issue: Compile error on Big Endian platform for Client/Server communication

**Module version 1.0.16**
2016-10-07

- Incorporated Bugzilla RfC 71047: Determining the length of the length field for variable size arrays
- Implemented usage of ImplementationDataType of (outermost) SwComponentType

**Module version 1.0.15**
2016-09-09

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.14**
2016-08-05

- Added support for memory partitioning of partitioned RTE
- ASCSOMEIPXF-311 Fixed known issue: Compilation error for a structure which holds a multidimensional array

**Module version 1.0.13**
2016-07-01

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.12**
2016-05-25

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.11**
2016-04-29

**Module version 1.0.10**
2016-04-01

- Added support of length fields for structures and fix size arrays

**Module version 1.0.9**
2016-02-05

- ASCSOMEIPXF-188 Fixed known issue: Erroneous value returned for client/server communication without arguments
- ASCSOMEIPXF-199 Fixed known issue: Wrong interface version 0 provided

**Module version 1.0.8**
2016-01-15

- ASCSOMEIPXF-177 Fixed known issue: Compilation fails for client/server communication without operation argument
- Added support of ApplicationDataTypes
- Incorporated Bugzilla RfC 68085: Clarification issues regarding the modeling of Variable-Size Array Data Type
- Incorporated Bugzilla RfC 67799: Fire-and-forget methods without parameters

**Module version 1.0.7**
2015-11-06

- Changed parameter name XfrmTransformationBswModuleEntryRef to XfrmTransformerBswModuleEntryRef (see AUTOSAR RfC #68531)
- Added support for specification of XfrmVariableDataPrototypeInstanceRef (multiple receivers)

**Module version 1.0.6**
2015-10-09

- Incorporated Bugzilla RfC 67586: SOME/IP Transformer does not support Message Type Notification

**Module version 1.0.5**
2015-09-18

- Incorporated Bugzilla RfC 68519: Rte_Cs_TransactionHandleType may have no members
- Incorporated Bugzilla RfC 68756: Clarification regarding the existence of the SOME/IP returnValue parameter when a server runnable has no application errors
- Implemented robust code generation by reporting a warning and ignoring problematic XfrmImplementationMappings

**Module version 1.0.4**
2015-06-19

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.3**
2015-04-24

- Added support of variable size arrays for basic data types
- Incorporated Bugzilla RfC 68035: Input parameters of scalar and enum types shall be passed by value
- Added support of client/server communication

**Module version 1.0.2**
2015-02-20

- Provided missing memory sections to function declarations

**Module version 1.0.1**
2015-01-07

- ASCSOMEIPXF-35 Fixed known issue: Erroneous serialization/deserialization of data elements following an array of structures

**Module version 1.0.0**
2014-10-02

- Initial AUTOSAR 4.2 version

### 3.3.2.2. New features

- No new features have been added since the last release.

#### 3.3.2.3. Elektrobit-specific enhancements

This chapter lists the enhancements provided by the module.

- Configurable data type for BufferLength

  Description:

  Module configuration parameter `XfrmBufferLengthType` enables the user to adjust the data type for parameter `BufferLength` of the transformer APIs.

  Configuring `XfrmBufferLengthType` to `UINT16` sets the data type of parameter `BufferLength` to `uint16`.

  Configuring `XfrmBufferLengthType` to `UINT32` sets the data type of parameter `BufferLength` to `uint32`.

  Rationale:

  This module shall be able to serialize and deserialize data with a size greater than 64 KiB.

- Configurable safety relevance of transformer

  Description:

  Module configuration parameter `XfrmIsSafetyTransformer` enables the user to explicitly state if the transformer is classified as safety relevant (ASILs) or not (QM).

  If parameter `XfrmIsSafetyTransformer` is enabled and configured to `TRUE`, the transformer is considered as a transformer classified for handling data with safety relevance (ASILs).

  If parameter `XfrmIsSafetyTransformer` is enabled and configured to `FALSE`, the transformer is considered as a transformer classified for handling data without safety relevance (QM).

  If parameter `XfrmIsSafetyTransformer` is disabled, the transformer is only considered as a transformer classified for handling data with safety relevance (ASILs), when a safety transformer with parameter `disableEndToEndCheck` set to `FALSE` is present within the same `DataTransformation`. Otherwise the transformer is considered as a transformer classified for handling data without safety relevance (QM).

  Rationale:

  This enhancement was introduced in order to provide a clear configuration semantics for the user.

- Memory partitioning

  Description:

  Module configuration parameter `XfrmOsApplicationRef` enables the user to assign the transformer to a dedicated memory partition.

If parameter `XfrmOsApplicationRef` is enabled and a valid reference to an `OsApplication` is provided, the global variables of the transformer are mapped to the memory partition to which the referenced `OsApplication` belongs.

If parameter `XfrmOsApplicationRef` is disabled, the global variables of the transformer are mapped to the default memory partition.

Rationale:

This enhancement was introduced in order to support the memory partitioning of the RTE.

- Different strategies for deriving the size of length fields of variable size arrays

Description:

Module configuration parameter `SomeIpXfEnableAR3462` enables the user to distinguish between different strategies for deriving the size of length fields of variable size arrays.

Configuring `SomeIpXfEnableAR3462` to `FALSE`, the default value, handles the size of the length fields for variable size array according to the AUTOSAR 4.2.1 specification. Meaning the size of the length field for variable size arrays is derived from the data type of the size indicator.

Configuring `SomeIpXfEnableAR3462` to `TRUE` uses the configured value of parameter `SOMEIPTransformationISignalProps.sizeOfArrayLengthFields` for the size of the length field for variable size arrays. If parameter `SOMEIPTransformationISignalProps.sizeOfArrayLengthFields` is not configured the default value of 32 bits is assumed.

Note:

If variable size array has assigned a TlvDataIdDefinition.id the configured value of parameter `SOMEIPTransformationISignalProps.sizeOfArrayLengthFields` is used. Regardless of the configured value of parameter `SomeIpXfEnableAR3462`.

Rationale:

This enhancement was introduced in order to give the user the option to choose between the old and new (improved) behavior and additionally avoid backwards incompatibilities. See also: AR-3462.

- SOME/IP Message Error Callouts

Description:

Module configuration parameter `SomeIpXfMessageErrorCallout` enables the user to control whether callout function declarations are generated for each configured inverse transformer or not.

Configuring `SomeIpXfMessageErrorCallout` to `TRUE` generates Callout declarations for each configured inverse transformer in order to provide return values that differ from `E_OK`. It is in the hands of the user to provide the callout function definitions.

Configuring `SomeIpXfMessageErrorCallout` to `FALSE`, the default value, does not generate any callout declaration for any configured inverse transformer.

The callouts follow the naming scheme `INVXF_NAME_MsgErrorCallout`, where `INVXF_NAME` is the transformer name (`XfrmImplementationMapping`).

Callout parameters:

- IN: `uint8`
- Return Value: `void`

**Example 3.1. Callout definition example:**

```
void SomeIpXf_Inv_TransformerId_MsgErrorCallout(uint8 RetVal)
{
  /* user-defined code */
}
```

Rationale:

This enhancement was introduced in order that the user can define its own message error counter behavior according to different return values.

- Configurable message type in the SOME/IP header for an autonomous error response

Description:

Module configuration parameter `SomeIpXfMessageTypeErrorResponse` enables the user to control which value is inserted into the message type field in the SOME/IP header for an autonomous error response on the server-ECU.

Configuring `SomeIpXfMessageTypeErrorResponse` to `RESPONSE`, the default value, inserts value 0x80 into the message type field.

Configuring `SomeIpXfMessageTypeErrorResponse` to `ERROR` inserts value 0x81 into the message type field.

Rationale:

This enhancement was introduced to overcome the ambiguity of the AUTOSAR specification. See also: AR-54502 and AR-103094.

- Tag-length-value (TLV) encoding

  Description:

  System model configuration container `TlvDataIdDefinition` enables the user to assign TLV identifiers to members of a structure and Client/Server method arguments.

  Rationale:

  This enhancement was introduced on request of the customers to provide the support for TLV encoding according to AUTOSAR release R19-11.

- Detection of duplicated Data IDs for method arguments that use tag-length-value (TLV) encoding

  Description:

  Module configuration parameter `SomeIpXfTlvArgumentUniqueDataIdCheck` enables the user to control whether deserialization is aborted or not if duplicated Data IDs for method arguments with TLV encoding of the same client/server deserialization API are encountered within the received serialized byte stream.

  Configuring `SomeIpXfTlvArgumentUniqueDataIdCheck` to `FALSE`, the default value, the deserialization continues even if a duplicated Data ID is encountered. If multiple method arguments with the same Data ID are present within the received serialized byte stream then the first one encountered is deserialized.

  Configuring `SomeIpXfTlvArgumentUniqueDataIdCheck` to `TRUE` increases the runtime performance but in turn aborts the deserialization with `E_SER_MALFORMED_MESSAGE` if a duplicated Data ID for method arguments with TLV encoding of the same client/server deserialization API is encountered. Only ids that are configured on the receiver side in the system model can be recognized as duplicated by this check.

  Rationale:

  This enhancement was introduced to overcome the ambiguity of the AUTOSAR specification and additionally to offer a runtime optimization for a use case where it is guaranteed that no duplicated method argument ids will be present within the received serialized byte stream. See also: AR-103433.

- Deserialization of method arguments that use tag-length-value (TLV) encoding in an arbitrary order

  Description:

  Module configuration parameter `SomeIpXfTlvArgumentReordering` enables the user to control whether reordering of the method arguments that use TLV encoding takes place during deserialization or not.

  Configuring `SomeIpXfTlvArgumentReordering` to `FALSE` aborts the deserialization with `E_SER_MALFORMED_MESSAGE` if the TLV method arguments within the received serialized byte stream are not in the same order as configured in the system model on the receiver side.

Configuring `SomeIpXfTlvArgumentReordering` to `TRUE`, the default value, increases the runtime performance but in turn deserializes the TLV method arguments even if they are not in the same order within the received serialized byte stream as they are configured in the system model on the receiver side.

Rationale:

This enhancement was introduced to offer a runtime optimization for a use case where it is guaranteed that method arguments within the received serialized byte stream are always in the same order as configured in the system model on the receiver side.

- Deserialization of structure members that use tag-length-value (TLV) encoding

Description:

Module configuration parameter `SomeIpXfTlvStructMemberOptions` enables the user to select which actions are performed during deserialization of structure members that use TLV encoding.

Configuring `SomeIpXfTlvStructMemberOptions` to `ENABLE_REORDERING_AND_UNIQUE_DATAID`, the default value, results in the worst runtime performance but in turn enables reordering and duplicated Data ID check. Structure members that use TLV encoding are deserialized even if they are not in the same order in the received serialized byte stream as they are configured in the system model on the receiver side. Additionally, the deserialization is aborted with `E_SER_MALFORMED_MESSAGE` if a duplicated Data ID for members of the same structure is encountered within the received serialized byte stream.

Configuring `SomeIpXfTlvStructMemberOptions` to `ENABLE_REORDERING_DISABLE_UNIQUE_DATAID` enables reordering and disables the duplicated Data ID check. Structure members that use TLV encoding are deserialized even if they are not in the same order in the received serialized byte stream as they are configured in the system model on the receiver side. Additionally, the deserialization continues even if a duplicated Data ID for members of the same structure is encountered. If multiple members of the same structure use the same Data ID within the received serialized byte stream then the last one encountered is deserialized.

Configuring `SomeIpXfTlvStructMemberOptions` to `DISABLE_REORDERING_AND_UNIQUE_DATAID` results in the best runtime performance but in turn disables both reordering and duplicated Data ID check. Deserialization is aborted with `E_SER_MALFORMED_MESSAGE` if members of a TLV structure within the received serialized byte stream are not in the same order as configured in the system model on the receiver side. Additionally, the deserialization continues even if a duplicated Data ID for members of the same structure is encountered. If multiple members of the same structure use the same Data ID within the received serialized byte stream then the first one encountered is deserialized.

Note:

Only Data IDs that are configured on the receiver side in the system model can be recognized as duplicated.

Rationale:

This enhancement was introduced to offer a runtime optimization for use cases where some or all of these actions during deserialization are not necessary. Additionally, because of the ambiguity of the AUTOSAR specification the user shall be able to select which actions are performed. See also: AR-103433.

- SomeIpXf_ExtractProtocolHeaderFields

  Description:

  Function to extract the relevant SOME/IP protocol header fields of the message and the type of the message result for client/server communication.

  Rationale:

  This enhancement was introduced to provide support for function SomeIpXf_ExtractProtocolHeaderFields according to AUTOSAR release R20-11.

- Description:

  The following Return Codes have been removed from Return Error codes. SOMEIPXF_E_UNKNOWN_SERVICE SOMEIPXF_E_UNKNOWN_METHOD

  Rationale:

  This enhancement was introduced to provide support for return codes. AUTOSAR release R21-11.

#### 3.3.2.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- Definition of Identifiers

  Description:

  The complete section 'Definition of Identifiers' and 'Reserved and special identifiers for SOME/IP and SOME/IP-SD is not supported.

  Requirements:

  SWS_SomeIpXf_00001 SWS_SomeIpXf_00025 SWS_SomeIpXf_00002 SWS_SomeIpXf_00005 SWS_SomeIpXf_00006 SWS_SomeIpXf_00007 SWS_SomeIpXf_00009 SWS_SomeIpXf_00010 SWS_SomeIpXf_00011 SWS_SomeIpXf_00130 SWS_SomeIpXf_00131 SWS_SomeIpXf_00132 SWS_SomeIpXf_00133 SWS_SomeIpXf_00134

- Strings of fixed length

  Description:

  The complete section 'Strings (fixed lengths)' is not supported

Requirements:

SWS_SomeIpXf_00053 SWS_SomeIpXf_00054 SWS_SomeIpXf_00055 SWS_SomeIpXf_00056 SWS_SomeIpXf_00057 SWS_SomeIpXf_00058 SWS_SomeIpXf_00059 SWS_SomeIpXf_00060 SWS_SomeIpXf_00017 (partly)

- Optional Parameters

  Description:

  The section 'Optional Parameters' is not supported

  Requirements:

  SWS_SomeIpXf_00076 SWS_SomeIpXf_00076_Deser SWS_SomeIpXf_00017 (partly)

- Strings of dynamic length

  Description:

  The section 'Strings (dynamic length)' is not supported

  Requirements:

  SWS_SomeIpXf_00076 SWS_SomeIpXf_00076_Deser SWS_SomeIpXf_00017 (partly)

- NoInitVal Bitfields

  Description:

  The Init values for Bitfields are not supported

  Requirements:

  SWS_SomeIpXf_00017 (partly) using NO initialization value for bitfields.

- Union / Variant

  Description:

  The complete section 'Union / Variant' is not supported.

  Requirements:

  SWS_SomeIpXf_00088 SWS_SomeIpXf_00098 SWS_SomeIpXf_00099 SWS_SomeIpXf_00017 (partly)

- Module Interlink Types Header File

  Description:

The module interlink types header file `SchM_SomeIpXfType.h` is included instead of the header file `SchM_SomeIpXf_Type.h` specified by SWS_SomeIpXf_00136.

Rationale:

This allows compatibility to the Rte which generates header file `SchM_SomeIpXfType.h`.

Requirements:

SWS_SomeIpXf_00136

- Postbuild

Description:

Postbuild is not supported.

Rationale:

There is no reason to implement post build functionality since the calling Rte module does not support it either.

Requirements:

SWS_SomeIpXf_00183

- Forward and Backward Compatibility

Description:

Forward and backward compatibility of parameters, arrays and structures is not supported,

except extended structures and arrays of fix size according to Bugzilla RfC 67775.

Requirements:

SWS_SomeIpXf_00016 (partly) SWS_SomeIpXf_00017 (partly) SWS_Xfrm_00048 (partly)

- Extended Production Errors

Description:

Extended production errors are not supported.

Requirements:

ECUC_Xfrm_00016 ECUC_Xfrm_00015 SWS_Xfrm_00070 SWS_Xfrm_00071

- Error Handling

Description:

Even though a hard error is returned, the output buffer of the serializer transformer APIs (SWS_SomeIpXf_00138, SWS_Xfrm_00036, SWS_Xfrm_00038, SWS_Xfrm_00040) will be changed. This means that the serialized data is written to the output buffer during serialization. If the hard error occurs the buffer where the serialized data is written to, will not be used.

Rationale:

Only one buffer is used for the serialization process which leads to better performance.

Requirements:

SWS_Xfrm_00051

- Request ID

Description:

Optional setting of the Request ID as mentioned in SWS_SomeIpXf_00106 is not implemented. Instead the behavior described in SWS_SomeIpXf_00024 is implemented. The Request ID is constructed of the Client ID and Session ID, which are chosen by the Rte and handed over to the SOME/IP transformer.

Requirements:

SWS_SomeIpXf_00106 (partly)

- Application Error with value 0 (E_OK)

Description:

Requirement SWS_SomeIpXf_00115 states on error codes in the range of 0x20 - 0x5e to adapt the value of the application error in adding 0x1F. This holds for application errors in the range of 0x01- 0x3F. The handling of possible error value 0 (E_OK) is not specified by any requirement. The possible error value 0 (E_OK) is handled untouched, i.e. directly written to the return code without adding 0x1F.

Rationale:

Incorporated Bugzilla RfC 76926: Contradiction between SWS_SomeIpXf_00107 and constr_1108.

Requirements:

SWS_SomeIpXf_00107 (partly)

- Development Error Tracer is not supported

Description:

- The Development Error Tracer DET is not supported due to safety aspects.

- The SomeIpXf module is maintained in two flavors. One at safety level according to ISO 26262 and another one at QM level. To keep this both close together, any handling by the DET module is removed. Instead it is handled by defensive programming techniques.

Rationale:

The function Det_ReportError() is only specified and implemented at QM level, not at any ASIL level.

Requirements:

SWS_SomeIpXf_00184, ECUC_Xfrm_00013_Conf

- SomeIpXf_Init() and SomeIpXf_DeInit() do not exist

Description:

The API service SomeIpXf_Init() and SomeIpXf_DeInit() do not exist as part of the SomeIpXf implementation.

Rationale:

The SomeIpXf needs no initialization of global data. Even if the necessity of an initialization routine might be the case in future, there would be a specific routine for each partition in order to support multi-core systems.

Requirements:

SWS_SomeIpXf_00181, SWS_SomeIpXf_00182

- Violation of VSMD rule Constr_3023

Description:

The attribute apiServicePrefix is mandatory for VSMDs derived from the CDD StMD. The attribute shall not be provided for VSMDs derived from any other StMDs. The rule is based on Constr_3023 from AUTOSAR_TPS_ECUConfiguration.pdf of 4.2.1 Release.

Effected node:

- StMD-Node: `/AUTOSAR/EcucDefs/Xfrm`

Rationale:

The module violates the second part of the constraint, but correctly, because SWS_SomeIpXf_00185 requires the definition of attribute apiServicePrefix. The discrepancy is handled by AUTOSAR with https://jira.autosar.org/browse/AR-109503 .

Requirements:

SWS_SomeIpXf_00185

- Violation of VSMD rule EcucSws_2038_2040_ASR41

  Description:

  If there is a SYMBOLIC-NAME-REFERENCE which points to another module, the rule EcucSws_2038_-2040_ASR41, which is based on requirements TPS_ECUC_02038 and TPS_ECUC_02040 from AUTOSAR_TPS_ECUConfiguration.pdf of 4.1 Release, always creates a violation.

  Effected nodes:

  - VSMD-Node: `Xfrm/XfrmImplementationMapping/XfrmDemEventParameterRefs/XFRM_E_MALFORMED_MESSAGE`
  - VSMD-Node: `Xfrm/XfrmImplementationMapping/XfrmOsApplicationRef`

  Rationale:

  Violation occurs due to an invalid behavior within the EB tresos Studio. No useful workaround available. Rule EcucSws_2038_2040_ASR41 shall be ignored.

  Requirements:

  ECUC_Xfrm_00016

- isDynamicLengthFieldSize is not supported

  Description:

  SOMEIPTransformationISignalProps.isDynamicLengthFieldSize set to true is not supported. Therefore, the transformer does not use wire types 5,6,7 for serializing complex types nor the size of the length field is set according to the wire type.

  Requirements:

  SWS_SomeIpXf_00270 (partly), SWS_SomeIpXf_00272, TPS_SYST_05017

- Definition of length field sizes on DataPrototype level is not supported

  Description:

  The definition of attributes sizeOfArrayLengthFields, sizeOfStructLengthFields, and sizeOfUnionLength-Fields on DataPrototype level is currently not supported. The attributes can be defined only on ISignal level.

  Requirements:

  constr_1654

- SOME/IP Transformer deserialize known TLV arguments/members with different but applicable wire type

  Description:

SOME/IP Transformer deserialize known TLV members/arguments and skip unknown TLV members/arguments with wire types 4, 5, 6 and 7 independent of the setting of SOMEIPTransformationISignalProps.isDynamicLengthFieldSize.

Rationale:

Requirement SWS_SomeIpXf_00273 does not clearly specify the meaning of the term 'handle' in the context of an deserializer which shall be able to 'handle' wire types 4, 5, 6 and 7 independent of the setting of SOMEIPTransformationISignalProps.isDynamicLengthFieldSize.

Requirements:

SWS_SomeIpXf_00273

#### 3.3.2.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- Limitation on Implementation Mapping

  Description:

  The short name of the implementation mapping has to be identical to the BswModuleEntry. I.e. the value of parameter XfrmImplementationMapping is the same as the referenced XfrmTransformerBswModuleEntryRef (or XfrmInvTransformerBswModuleEntryRef).

  As a consequence, either the XfrmTransformerBswModuleEntryRef or the XfrmInvTransformerBswModuleEntryRef might be referenced, not both.

  Additionally, the recommended optimization on the transformer chain can not be applied the way specified, since it is not possible to reference the same BswModuleEntry for different ImplementationMappings.

  Rationale:

  This avoids usage of an X-Path expression to extract data from the tresos DB.

  Requirements

  SWS_Xfrm_00054, SWS_Xfrm_00101 (Incorporated by Bugzilla RfC 67799), SWS_Xfrm_00055 and SWS_Xfrm_00056

- Restrictions on extensible structures (structures with length fields and without TLV encoding)

  Description:

Extensible structures containing a variable size array of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE are not supported.

- Restrictions on structures with tag-length-value (TLV) encoding

  Description:

  Structures with TLV encoding containing a variable size array member of profile VSA_SQUARE or VSA_-RECTANGULAR are not supported at any nested level.

- Restrictions on standard arrays (fixed size arrays without length fields)

  Description:

  The ImplementationDataType of category ARRAY containing a composite ImplementationDataTypeElement in the role of a subElement of category STRUCTURE is not supported. Instead, contained structure shall be created by modeling ImplementationDataTypeElement of category TYPE_REFERENCE that in turn refers to a ImplementationDataType of category STRUCTURE.

  Standard multidimensional arrays containing variable size arrays as elements are not supported. This holds for each variable size array profile type.

- Restrictions on extensible arrays (fixed size arrays with length fields)

  Description:

  The ImplementationDataType of category ARRAY containing a composite ImplementationDataTypeElement in the role of a subElement of category STRUCTURE is not supported. Instead, contained structure shall be created by modeling ImplementationDataTypeElement of category TYPE_REFERENCE that in turn refers to a ImplementationDataType of category STRUCTURE.

  Extensible arrays containing a variable size array of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE are not supported.

- Restrictions on variable size arrays

  Description:

  The payload of variable size arrays of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE containing an extensible array (fixed size array with length fields) at any nested level is not supported.

  The payload of variable size arrays of profile VSA_SQUARE or VSA_RECTANGULAR containing a standard structure (structure without length fields) at any nested level is not supported.

  The payload of variable size arrays of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE containing an extensible structure (structure with length fields) at any nested level is not supported.

The payload of variable size arrays of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE containing a structure with TLV encoding at any nested level is not supported.

The payload of variable size arrays of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE containing a standard array (fixed size array without length fields) at any nested level is not supported.

The payload of variable size arrays of profile VSA_SQUARE, VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE containing a variable size array of any profile at any nested level is not supported.

If a size indicator of a variable size array of profile VSA_RECTANGULAR or VSA_FULLY_FLEXIBLE is set to zero for a respective dimension all following size indicator values for the lower level dimensions get handled as set to zero.

The number of dimensions for a variable size array of profile VSA_FULLY_FLEXIBLE are considered as number of nested variable size arrays (i.e. VSA_SQUARE with 3 dimensions of structs which holds a VSA_FULLY_FLEXIBLE of 10 dimensions results in a number of nested variable size arrays of 11 which is not supported).

- Restrictions on Bitfields

  Description:

  Only standard structures (without length fields) can hold Bitfields as members.

  Bitfields not supported on big endian platforms (CPU_BYTE_ORDER configured with HIGH_BYTE_FIRST)

- Restrictions on method arguments with tag-length-value (TLV) encoding

  Description:

  Method arguments with TLV encoding that are associated with data type variable size array of profile VSA_SQUARE or VSA_RECTANGULAR are not supported.

#### 3.3.2.6. Open-source software

SomeIpXf does not use open-source software.

## 3.3.3. Xfrm module release notes

- AUTOSAR R4.0 Rev 3
- AUTOSAR SWS document version: 0.0.0

- Module version: 1.0.39.B567464
- Supplier: Elektrobit Automotive GmbH

#### 3.3.3.1. Change log

This chapter lists the changes between different versions.

**Module version 1.0.39**
2022-10-12

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.38**
2022-03-09

- Fixed invalid consideration of configuration container EndToEndTransformationComSpecProps for sending transformers

**Module version 1.0.37**
2022-01-28

- ASCXFRM-439, ASCCOMXF-639 Fixed known issue: Code generation for multiple receivers results in an error

**Module version 1.0.36**
2021-11-10

- ASCXFRM-434, ASCCOMXF-628 Fixed known issue: Safety Com transformer forwards undersized buffer length to the E2EXf

**Module version 1.0.35**
2021-10-08

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.34**
2021-09-17

- Improved error message for the invalid ImplementationDataType reference of the container SwDataDefProps
- Added support for SomeIpXf_ExtractProtocolHeaderFields API

**Module version 1.0.33**
2021-04-09

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.32**
2021-03-05

- Updated preprocessor include guards to be PC-lint compatible

**Module version 1.0.31**
2021-02-12

- Implemented support for data types with identifier (tag/length/value)

**Module version 1.0.30**
2020-07-31

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.29**
2020-05-22

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.28**
2020-01-24

- ASCXFRM-365 Fixed known issue: Variable Size Arrays are serialized to the wrong array type

**Module version 1.0.27**
2019-10-11

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.26**
2019-03-22

- Enhanced support of parameter disableEndToEndCheck to single receivers

**Module version 1.0.25**
2019-02-15

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.24**
2018-10-26

- Prevent transformers from generating invalid BSWMD.arxml when configuration of XfrmImplementation-Mapping is empty

**Module version 1.0.23**
2018-07-27

- Align memory mapping with safety options according to AUTOSAR 4.3 MetaModel

**Module version 1.0.22**
2018-06-22

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.21**
2018-05-25

- ASCXFRM-309 Fixed known issue: Compile error in generated SomeIpXf_Api_gen.h

**Module version 1.0.20**
2018-03-16

- Updated the compatibility check interface

**Module version 1.0.19**
2017-12-15

- Implemented support for configurable type of BufferLength
- Added XfrmIsSafetyTransformer configuration parameter
- Updated the supported datatypes to AUTOSAR release 4.2.1

**Module version 1.0.18**
2017-09-22

- Add safe transformer condition check

**Module version 1.0.17**
2017-07-28

- Implemented usage of ApplicationDataType of (outermost) SwComponentType

**Module version 1.0.16**
2017-06-02

- Added support for Safe ComXf

**Module version 1.0.15**
2017-05-05

- Improved gathering of computational methods
- Added support for variable size array profile of type fully flexible

**Module version 1.0.14**
2017-03-31

- Incorporated Bugzilla RfC 69896: Execution of Transformer chain in case of unqueued communication when no data is available
- Incorporated Bugzilla RfC 68623: Insufficient specification of autonomous error response

**Module version 1.0.13**
2017-03-03

- ASCXFRM-150 Fixed known issue: Compile error occurs when ImplementationDataType of the outermost CompositionSwComponent differs from atomic software component for client/server communication

**Module version 1.0.12**
2017-02-03

- Enable Xfrm users to gather computational methods from the abstracted data type model by providing a reference to SwDataDefProps

**Module version 1.0.11**
2017-01-05

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.10**
2016-12-02

- ASCXFRM-144 Fixed known issue: Compile error occurs when ImplementationDataType of the outermost CompositionSwComponent differs from atomic software component for sender/receiver communication

**Module version 1.0.9**
2016-11-04

- Incorporated Bugzilla RfC 70485: Missing configuration parameter XfrmVersionInfoApi

**Module version 1.0.8**
2016-10-07

- Implemented usage of ImplementationDataType of (outermost) SwComponentType

**Module version 1.0.7**
2016-09-09

- Internal module improvement. This module version update does not affect module functionality

**Module version 1.0.6**
2016-07-01

- ASCXFRM-82 Fixed known issue: Generic SwBaseTypes when referenced by ImplementationDataType lead to incorrect generated configuration items

**Module version 1.0.5**
2016-05-25

- Provided error message for usage of unsupported basic data types

**Module version 1.0.4**
2016-04-29

- Added support for generic SwBaseType

**Module version 1.0.3**
2016-04-01

- Incorporated Bugzilla RfC 67775: SOME/IP Transformer uses wrong length of length field (extensible fixed size arrays)

**Module version 1.0.2**
2016-02-05

- Addded support of ApplicationDataType
- Incorporated Bugzilla RfC 68085: Clarification issues regarding the modeling of Variable-Size Array Data Type

**Module version 1.0.1**
2015-11-06

- Changed parameter name XfrmTransformationBswModuleEntryRef to XfrmTransformerBswModuleEntryRef (see AUTOSAR RfC #68531)
- Added support for specification of XfrmVariableDataPrototypeInstanceRef (multiple receivers)

**Module version 1.0.0**
2015-10-09

- Initial release of Transformer library

#### 3.3.3.2. New features

- No new features have been added since the last release.

#### 3.3.3.3. Elektrobit-specific enhancements

This chapter lists the enhancements provided by the module.

- This module provides no Elektrobit-specific enhancements.

#### 3.3.3.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- For this module no deviations are known.

#### 3.3.3.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- For this module no limitations are known.

#### 3.3.3.6. Open-source software

Xfrm does not use open-source software.

# 4. ACG8 Transformers user guide

## 4.1. Overview

This user guide describes the concepts and the configuration of the modules:

- `ComIpXf` COM Based Transformer
- `SomeIpXf` SOME/IP Transformer

To understand the basic concepts of the transformer modules, see Section 4.2, “Background information”.

To use data transformation in your project, see Section 4.3, “Using data transformation support”.

For instructions how to configure the modules, see:

- Section 4.4, “ComXf module user guide”
- Section 4.5, “SomeIpXf module user guide”

## 4.2. Background information

This chapter enables you to understand the basic concepts of the AUTOSAR transformer modules. If you are familiar with these concepts already, you may want to skip this chapter and proceed to the instruction chapters.

### 4.2.1. Functional overview

This section provides a brief overview of the data transformation concept as defined by AUTOSAR. You can find detailed information in the AUTOSAR 4.2 documents.

Data transformation is a new communication paradigm in AUTOSAR 4.2 that provides the following:

- Efficient communication of large data elements of complex types
- Simplified configuration of additional safety or security mechanisms

The basic idea is that, in the transmission path, a data element is first converted to a byte array. After that, you can use additional transformer modules to process the byte array to add safety or security functionality. In the

reception path, inverse functions are called in reverse order so that first safety checks or security transformations are applied and finally the byte array is transformed back to the data element.

In this context, the first transformer on the transmission path is called a *serializing* transformer, as it *serializes* a data element to a byte array or *de-serializes* the byte array back to a data element. Further transformers just work on the byte array, i.e. have a byte array as input and as output. Here, transformers can be of the following classes:

- A *safety* transformer ensures that the input byte array is not unintentionally modified during transmission. To do this, a CRC value is appended to the original byte array.
- A *security* transformer ensures that the data is not intentionally modified by authenticating and encrypting the transmitted data. Additionally, the data may be encrypted to protect against eavesdropping.
- A *custom* transformer is used for any other functionality.

The data transformation scenario is highly configurable to your needs and allows these options:

- Byte-wise data serialization according to the scalable service-oriented middleware over IP (SOME/IP) specification or data serialization according to the `Com` module configuration.

  Byte-wise data serialization according to SOME/IP means that, during data serialization, each single data element of the complex data type is written to a specific byte offset in the serialized data array and thus starts at a byte boundary. This kind of serialization is implemented by the `SomeIpXf` module. Bit shifting and packing is not required, making this kind of transformation fast. However, this results in more data and might not necessarily be compatible with other ECUs on the network. If the SOME/IP transformer is used, the data transferred can only be received by other ECUs that also use SOME/IP transformation. This is incompatible with older ECUs that use the `Com` module for data transfer or ECUs that use the `ComXf` module for data serialization.

  Serialization according to the `Com` module configuration means that the complex data element is packed into the byte array as configured in the `Com` module, i.e. based on the configured bit offset and length of each signal. The effect is that the serialized data element has the same memory layout as it would have without serialization and if it were then transmitted using the `Com` module. This results in identical representation of the data on the network. This kind of serialization is implemented by the `ComXf` module.
- Transformer chains and transformer fan-out

  If more than one transformer is used, i.e. if additional transformers are attached to the serializing transformer, this is called a *transformer chain*. The output of one transformer is then provided as input to the next transformer.

  The first transformer in the chain is a serializing transformer that converts the data element to a byte array. Other transformers work on the serialized data stream and produce a serialized data stream with either changed data, e.g. in the case of a security transformer. Or, with appended data, e.g. in the case of a safety transformer. When you use transformer chains, there is room for optimization if data is sent to more than one recipient (fan-out). In this case, the `Rte` ensures that transformer functions are called if necessary.

**NOTE** **Data transformation chains in the Rte**

For more information on data transformation chains in the `Rte` module, see the EB tresos AutoCore Generic 8 RTE product documentation.

- Data serialization together with end-to-end protection by a safety transformer

  You can use both serializing transformers (`ComXf` and `SomeIpXf`) together with an end-to-end protection transformer (`E2EXf`). This means that data is first serialized, then the end-to-end checksum is calculated and appended to the serialized data stream. From a software component's point of view there is no difference between sending protected and sending unprotected data. It suffices to model the data exchange accordingly and include the `E2EXf` module in the transformer chain to have the data end-to-end protected. Thus the software component can be relocated among ECUs without any modifications in the software component's implementation.

- Buffer handling

  The system description can model the buffer consumption of a transformer as *out-of-place* or *in-place*. Out-of-place means that the input pointer to the transformer points to another memory location than the output pointer and the transformer may safely write to the output pointer without overwriting the input data. This case requires more memory but is required for some types of transformers. The serializing transformer for example is always an out-of-place transformer. In-place buffer handling means that the input and the output pointer of the transformer call point to the same memory location. It depends on the type of transformer if this type of buffer handling is possible. For example, for some profiles of end-to-end protection the original data is not changed and just the checksum is appended, so this is possible.

## 4.2.2. Data transformation use cases

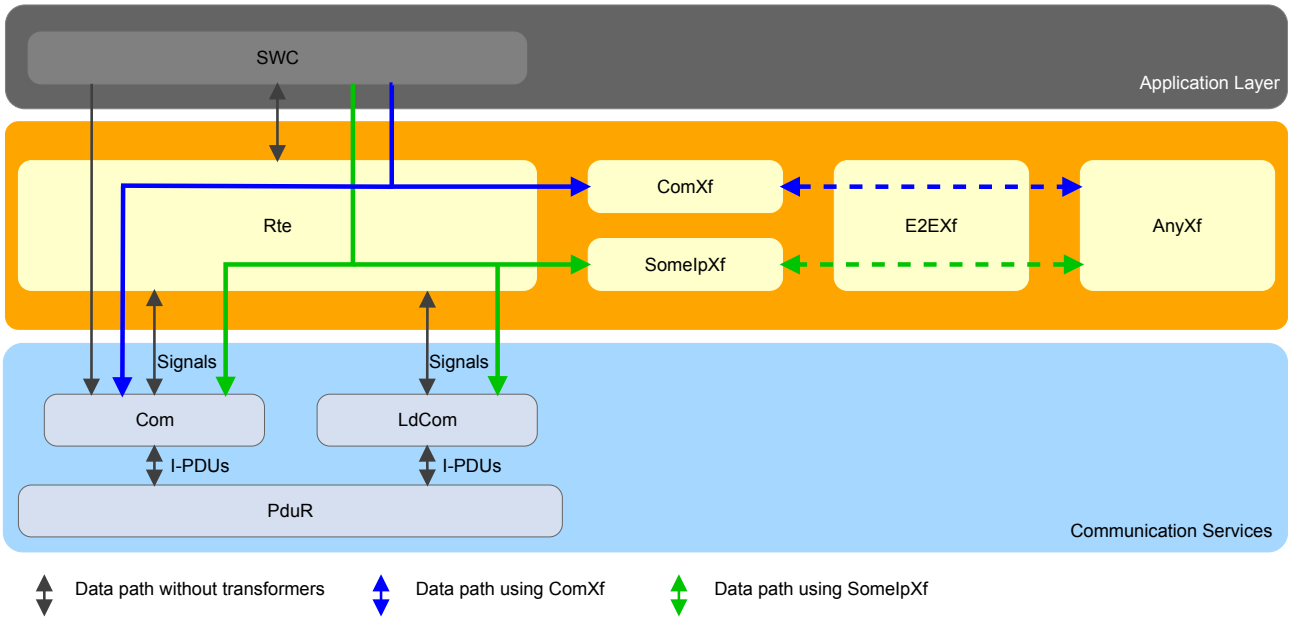


Figure 4.1. Use cases of data transmission

This section explains the use cases and the benefits of data transformation. The following use cases are possible:

- The traditional use case: SWC - `Rte` - `Com`

  This is the traditional use case of data transmission without any data transformation and is just mentioned for comparison. The `Rte` updates each signal of the complex data element and then requests transmission as a signal group. Updating each signal involves bit shifting and packing for each member of the data element.

- SWC - `Rte` - `SomeIpXf` - `Com`

  In this use case, the `SomeIpXf` serializes a complex data element to a byte array. This byte array is then transmitted via the `Com` module. This is particularly efficient with respect to execution time for large and complex data elements because the `SomeIpXf` simply writes each element of the data type to a separate byte and there is no need for bit packing and shifting. Using the `Com` module in this case has the advantage that other `Com` features can still be used, e.g. deadline monitoring. There is no additional ROM consumption as would be the case if the `LdCom` module and the `Com` module were used in parallel.

- SWC - `Rte` - `SomeIpXf` - `LdCom`

  Similarly to above, the `SomeIpXf` serializes the data to a byte array but this is then transmitted via the `LdCom` module. This reduces execution time and in addition may save ROM if the `Com` module is not required at all, since the `LdCom` provides only a minimum functionality compared to the `Com` module.

- SWC - `Rte` - `ComXf` - `Com`

  In this use case, the `ComXf` serializes a complex data element to a byte array, which is then transmitted via the `Com` module. This use case is still more efficient than the transmission of the data element as a signal group via the `Com` module. However, as the `ComXf` serializes according to the `Com` configuration and does similar bit packing and shifting, it has a higher execution time than the `SomeIpXf`. One important benefit of this use case is that data on the network is identical to the traditional signal-based case and thus ECUs in this use case can be added to existing networks.

- SWC - `Rte` - `SomeIpXf`/`ComXf` - `E2EXf` - `Com`/`LdCom`

  In this use case, a complex data element is first serialized using `SomeIpXf` or `ComXf`, then the serialized byte array is end-to-end protected and a sequence counter and checksum are appended to the byte array. The byte array is then transmitted over `Com` or `LdCom`. Besides the advantages mentioned above, this use case also has an important benefit regarding the configuration. The system description already models the data serialization and the end-to-end protection and thus no additional configuration effort is required to have end-to-end protected communication. No changes in the SWC implementation are required when they are relocated among ECUs.

### 4.2.3. AUTOSAR modules for data transformation

This section lists the AUTOSAR modules that are involved in data transformation and briefly explains their main functionality as far as it is relevant for data transformation. You can find detailed information in the AUTOSAR software specification documents of AUTOSAR 4.2. For more information on the following AUTOSAR modules, see the respective product-specific documentation.

- `Rte`

  The `Rte` implements a mapping of complex data elements to a byte array by data transformation. Core functionality is provided by a transformer dispatcher. This dispatcher calls the transformer functions of the transformer chain that is associated with the mapping in the order specified in the system description. The transformed byte array is then passed to `Com` or `LdCom` module for transmission. In the reception path, a byte array is received and indicated to the `Rte`. The `Rte` then uses the transformer dispatcher to execute the transformer chain in reverse order to retrieve the data element received.

- `SomeIpXf`

  The `SomeIpXf` transformer is a serializing transformer that transforms a data element to a byte array according to the SOME/IP specification. It provides APIs for serialization (in the transmission path) and de-serialization (in the reception path).

- `ComXf`

The `ComXf` transformer is a serializing transformer that transforms a data element to a byte array according to the `Com` module configuration. It provides APIs for serialization (in the transmission path) and deserialization (in the reception path).

- `E2EXf` and other non-serializing transformer modules

  You can add safety, security, and custom transformers as successors to a serializing transformer for specific transformation tasks. They provide APIs for transformation and inverse transformation of one byte array to another byte array calculating end-to-end checksums or message authentication codes.

- `Com`

  The `Com` module provides the following API functions:

  - `Com_SendDynSignalArray()` and `Com_ReceiveDynSignalArray()` to transmit or receive data that was serialized using the `ComXf` module.
  - `Com_SendDynSignal()` and `Com_ReceiveDynSignal()` to transmit or receive data that was serialized using the `SomeIpXf` module.

  Additionally, you can enable other `Com` functionality, e.g. deadline monitoring.

- `LdCom`

  The `LdCom` module provides the following API functions for sending and receiving byte arrays:

  - `LdCom_IfTransmit()`, `LdCom_RxIndication()`, `LdCom_TriggerTransmit()`, and `LdCom_TxConfirmation()` to transmit or receive data that fits into a single frame of the underlying network, i.e. IF-API.
  - `LdCom_StartOfReception()`, `LdCom_TpTransmit()`, `LdCom_CopyRxData()`, `LdCom_CopyTxData()`, `LdCom_TpTxConfirmation()`, and `LdCom_TpRxIndication()` to transmit or receive data that does not fit into a single frame of the underlying network and thus needs segmentation, i.e. TP-API.

  It has no function other than to transfer a byte array signal into a PDU. If no other functionality is required, then using `LdCom` is more efficient and requires less memory than using the `Com` module.

# 4.3. Using data transformation support

This section describes the steps to use data transformation in your project.

## 4.3.1. Adding data transformation to the system description

To enable data transformation, add the following elements to the system description:

- Definitions of DATA-TRANSFORMATION-SET

  DATA-TRANSFORMATION-SET describes a transformer chain. A transformer chain consists of one or several transformers where each transformer has a specific TRANSFORMATION-TECHNOLOGY which models transformer properties such as type of transformation and buffer requirements.

- Data mappings between complex data elements and byte array signals

  In the data transformation use case, a complex data element shall be transmitted as a byte array. Therefore, a mapping between the complex data element and a byte array signal has to be modelled. Such a mapping is only possible for signals referencing a DATA-TRANSFORMATION-SET.

## 4.3.2. Configuring data transformation modules using EB tresos Studio

Follow these steps to configure data transformation using EB tresos Studio:

- Add the modules you want to configure to your project.

  In particular add the Rte, a serializing transformer (e.g. SomeIpXf) and either Com or LdCom.

- Import the system description and run the **Import ECU Configuration** unattended wizard.

  Note that the **Import ECU Configuration** unattended wizard uses a complex rule set to determine if a signal is to be created in the Com module or the LdCom module. The LdCom is only used for a particular signal if this module is available and no other functionality is modeled in the system description for that signal that would require the Com module.

- Configure the serializing transformer, e.g. the SomeIpXf module:
  - Add a container XfrmImplementationMapping for each transformer and signal, i.e. configure this mapping for each transformer that is applied to a signal. For example, if two different signals are first serialized by SomeIpXf, then transformed by E2EXf, they need four mappings. You must map both signals to the SomeIpXf and also to the E2EXf.
  - Enable the reference XfrmTransformationBswModuleEntry for the transmission path or XfrmInvTransformationBswModuleEntry for the reception path. You must enable exactly one reference.
  - Generate the BSWMD. You can only reference the transformer functions after the BSWMD is generated.
  - Configure the reference XfrmTransformationBswModuleEntry for the transmission path or XfrmInvTransformationBswModuleEntry for the reception path. Choose the correct function references. Note that the name of the XfrmImplementationMapping container also determines the name of the generated function.

Different mappings might refer to the same BSW module definition. Due to the fact that each mapping generates its own BSW module definition entry, some entries are then unused.

- Configure the `XfrmTechnology` and reference the correct transformer technology from the system description.
- Configure the `XfrmSignal` and reference the correct signal from the system description.
- If the respective signal is used by more than one receiving software component, configure the `XfrmVariableDataPrototypeInstanceRef` as well in case dedicated transformer chains shall be used by the different receiving software components.

The above steps are automated by the **Calculate Service Needs** unattended wizard for the `SomeIpXf`/`ComXf` `XfrmImplementationMapping`.

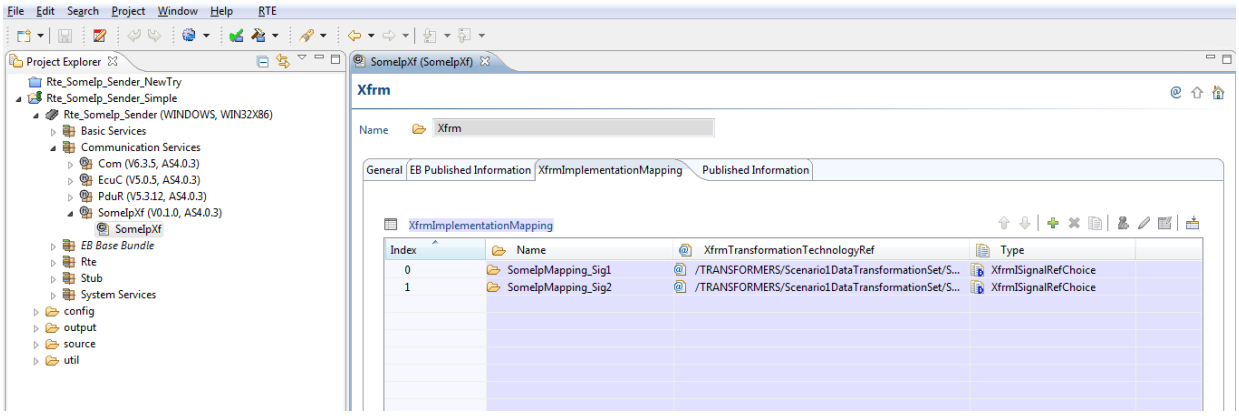


Figure 4.2. Configuring `SomeIpXf`

# 4.4. ComXf module user guide

## 4.4.1. Overview

This user guide describes the `ComXf` module. From this user guide you learn about the basic functionality of the `ComXf`. You also learn which related modules are necessary to configure the `ComXf` module. The `ComXf` module reference provides further information on configuring the `ComXf` itself.

Note that this user guide is intended for readers who have good knowledge of AUTOSAR and about the purpose of the `ComXf`. The information provided here helps you to integrate the `ComXf` in your AUTOSAR project.

- Section 4.4.2, “Background information” provides an overview of the basic functionality of the `ComXf`.

- Section 4.4.3, “Configuring ComXf” provides information on related modules that are needed in order to configure the `ComXf`.
- Section Section 4.4.4, “`ComXf` integration notes” provides notes for the integration of the `ComXf` module into your project.
- For details on configuring the `ComXf` itself, see the parameter descriptions provided in the `ComXf` module reference Chapter 5, “ACG8 Transformers module references“.

## 4.4.2. Background information

The general concept about data transformation and large data transfer is described in Section 4.2, “Background information”.

The `ComXf` is a serializing transformer that implements data serialization according to the `ComXf` specification. For more information, see the Specification of COM Based Transformer, AUTOSAR 4.2.1.

It is used together with the modules `Rte` and `Com` to implement data serialization according to a fixed communication matrix with packed data representations (`ComSignalGroups`) for the following communication paradigm:

- Sender/receiver communication

  `ComXf` provides APIs to serialize and deserialize complex data elements to byte arrays for sender/receiver communication. Its API functions are called by the `Rte`.

## 4.4.3. Configuring ComXf

To configure the `ComXf` module, add the module to your project using EB tresos Studio. Parameter descriptions are provided to guide the configuration. You find these in the module references section of this document. You also find these in the parameter description in EB tresos Studio.

The `ComXf` transformer has a mechanism to generate transformations with QM classification. The rationale is to allow the use of an enhanced non-safety certified feature set, e.g. the use of the `Com` module. The optional parameter `XfrmIsSafetyTransformer` determines if a related transformation, which is represented by `XfrmImplementationMapping`, shall be used for data with safety or QM classification.

If a transformation is used to transform data with safety classification, all of the following conditions shall apply:

- `XfrmIsSafetyTransformer` shall be enabled **AND**
- `XfrmIsSafetyTransformer` shall be set to `true`

If a transformation is used to transform data without safety classification, any value is allowed.

**NOTE**

**Significance of `XfrmIsSafetyTransformer` parameter**

If the `XfrmIsSafetyTransformer` parameter is disabled, the generator determines the classification of a transformation by its own heuristic. The transformation gets a safety classification if all of the following conditions apply:

- The `E2EXf` module is part of the (de-)transformer chain.
- The `disableEndToEndCheck` attribute in the `EndToEndTransformationComSpecProps` of the `SystemTemplate` is set to `false`.

In any other case, the transformation gets a QM classification.

**NOTE**

**Consistency of `XfrmIsSafetyTransformer`**

The `XfrmIsSafetyTransformer` parameter shall have the same value for all transformations of the same partition. If this is not the case, all transformations of the partition are promoted to safety classification.

To use the `ComXf` module, you must configure additional modules as outlined below:

- ACG8 RTE: The `ComXf` module provides API functions that are called by the `Rte` module in EB tresos AutoCore Generic 8 RTE.
- ACG8 COM Services: The `Rte` communicates the serialized data resulting from the `ComXf` module to the `Com` module, which is part of EB tresos AutoCore Generic 8 COM Services.

### 4.4.4. `ComXf` integration notes

You find module-specific information about exclusive areas, production errors, and memory mapping in the module-specific integration notes in the module references chapter of this document. See Chapter 5, “ACG8 Transformers module references“ subsection `Integration notes` in each module.

## 4.5. SomeIpXf module user guide

### 4.5.1. Overview

This user guide describes the `SomeIpXf` module. From this user guide you learn about the basic functionality of the `SomeIpXf`. You also learn which related modules are necessary to configure the `SomeIpXf` module. The `SomeIpXf` module reference provides further information on configuring the `SomeIpXf` itself.

Note that this user guide is intended for readers who have good knowledge of AUTOSAR and about the purpose of the `SomeIpXf`. The information provided here helps you to integrate the `SomeIpXf` in your AUTOSAR project.

- Section 4.5.2, "Background information" provides an overview of the basic functionality of the `SomeIpXf`.
- Section 4.5.3, "Configuring SomeIpXf" provides information on related modules that are needed in order to configure the `SomeIpXf`.
- Section Section 4.5.4, " `SomeIpXf` integration notes " provides notes for the integration of the `SomeIpXf` module into your project.
- For details on configuring the `SomeIpXf` itself, see the parameter descriptions provided in the `SomeIpXf` module reference Chapter 5, "ACG8 Transformers module references".

## 4.5.2. Background information

The general concept about data transformation and large data transfer is described in Section 4.2, "Background information".

The `SomeIpXf` is a serializing transformer that implements data serialization according to the SOME/IP specification. For more information, see the Specification of SOME/IP Transformer, AUTOSAR 4.2.1.

It is used together with the modules `Rte` and `LdCom` to enable data transformation for the following communication paradigms:

- Sender/receiver communication

  `SomeIpXf` provides APIs to serialize and deserialize complex data elements to byte arrays for sender/receiver communication. Its API functions are called by the `Rte`.
- Client/server communication

  `SomeIpXf` provides APIs to serialize and deserialize complex data elements to byte arrays for client/server communication. Its API functions are called by the `Rte`.

## 4.5.3. Configuring SomeIpXf

To configure the `SomeIpXf` module, add the module to your project using EB tresos Studio. Parameter descriptions are provided to guide the configuration. You find these in the module references section of this document. You also find these in the parameter description in EB tresos Studio.

The `SomeIpXf` transformer has a mechanism to generate transformations with QM classification. The rationale is to allow the use of an enhanced non-safety certified feature set, e.g. the support of advanced data types. The

optional `XfrmIsSafetyTransformer` parameter determines if a related transformation, which is represented by `XfrmImplementationMapping`, shall be used for data with safety or QM classification.

If a transformation is used to transform data with safety classification, all of the following conditions shall apply:

- `XfrmIsSafetyTransformer` shall be enabled **AND**
- `XfrmIsSafetyTransformer` shall be set to `true`

If a transformation is used to transform data without safety classification, any value is allowed.

**NOTE**

**Significance of `XfrmIsSafetyTransformer` parameter**

If the `XfrmIsSafetyTransformer` parameter is disabled, the generator determines the classification of a transformation by its own heuristic. The transformation gets a safety classification if all of the following conditions apply:

- The `E2EXf` module is part of the (de-)transformer chain.
- The `disableEndToEndCheck` attribute in the `EndToEndTransformationComSpecProps` of the `SystemTemplate` is set to `false`.

In any other case, the transformation gets a QM classification.

**NOTE**

**Consistency of `XfrmIsSafetyTransformer`**

The `XfrmIsSafetyTransformer` parameter shall have the same value for all transformations of the same partition. If this is not the case, all transformations of the partition are promoted to safety classification.

To use the `SomeIpXf` module, you must configure additional modules as outlined below:

- ACG8 RTE: The `SomeIpXf` module provides API functions called from the `Rte` module in EB tresos AutoCore Generic 8 RTE.
- EB tresos AutoCore Generic 8 LDCOM: The `Rte` communicates the serialized data resulting from the `SomeIpXf` module to the `LdCom` module, which is part of EB tresos AutoCore Generic 8 LDCOM.

## 4.5.4. `SomeIpXf` integration notes

You find module-specific information about exclusive areas, production errors and memory mapping in the module-specific integration notes. You find the module-specific integration notes in the module references chapter of this document. See Chapter 5, “ACG8 Transformers module references“ sub-section `Integration notes` in each module.

# 5. ACG8 Transformers module references

## 5.1. Overview

This chapter provides module references for the ACG8 Transformers product modules. These include a detailed description of all configuration parameters. Furthermore this chapter lists the application programming interface with all data types, constants and functions.

The content of the sections is sorted alphabetically according the EB tresos AutoCore Generic module names.

For further information on the functional behavior of these modules, refer to the chapter ACG8 Transformers user's guide.

### 5.1.1. Notation in EB module references

EB notation may differ from the AUTOSAR standard notation in the software specification documents (SWS). This section describes the notation of *default value* and *range* fields in the EB module references.

#### 5.1.1.1. Default value of configuration parameters

If there is no default value specified for a parameter, the default value field is omitted to prevent ambiguity with parameters that have `--` as default values.

Example: The parameter `BswMCompuConstText` of the `BswM` module of EB tresos AutoCore Generic 8 Mode Management has no default value field, therefore it is omitted.

#### 5.1.1.2. Range information of configuration parameters

The range of a configuration parameter contains an upper and a lower boundary. However, in special cases the range of allowed values can be computed by means of an XPath function that is evaluated at configuration time. An XPath function can either be a standard `xpath:<function>()` or a custom `cxpath:<function>()` function. The range of a configuration parameter may be computed based on other configuration parameters that are referenced from the XPath function. For more information on custom XPath functions, see section *Custom XPath Functions API* of the EB tresos Studio developer's guide.

Example: The parameter `BswMCompuConstText` of the `BswM` module of EB tresos AutoCore Generic 8 Mode Management has the custom XPath function `cxpath:getCompuMethodsVT()` in the range field which provides the allowed values.

# 5.2. ComXf

## 5.2.1. Configuration parameters

| Containers included | | |
|---|---|---|
| **Container name** | **Multiplicity** | **Description** |
| CommonPublishedInformation | 1..1 | **Label:** Common Published Information<br>Common container, aggregated by all modules. It contains published information about vendor and versions. |
| PublishedInformation | 1..1 | **Label:** EB Published Information<br>Additional published parameters not covered by CommonPublishedInformation container. |
| XfrmGeneral | 1..1 | Contains the general configuration parameters of the module. |
| XfrmImplementationMapping | 1..n | 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. |

| Parameters included | |
|---|---|
| **Parameter name** | **Multiplicity** |
| IMPLEMENTATION_CONFIG_VARIANT | 1..1 |

| Parameter Name | IMPLEMENTATION_CONFIG_VARIANT |
|---|---|
| **Label** | Config Variant |
| **Multiplicity** | 1..1 |
| **Type** | ENUMERATION |
| **Default value** | VariantPreCompile |
| **Range** | VariantPreCompile |

### 5.2.1.1. CommonPublishedInformation

| Parameters included | |
|---|---|
| **Parameter name** | **Multiplicity** |

| Parameters included | |
|---|---|
| ArMajorVersion | 1..1 |
| ArMinorVersion | 1..1 |
| ArPatchVersion | 1..1 |
| SwMajorVersion | 1..1 |
| SwMinorVersion | 1..1 |
| SwPatchVersion | 1..1 |
| ModuleId | 1..1 |
| VendorId | 1..1 |
| Release | 1..1 |

| Parameter Name | ArMajorVersion |
|---|---|
| Label | AUTOSAR Major Version |
| Description | Major version number of AUTOSAR specification on which the appropriate implementation is based on. |
| Multiplicity | 1..1 |
| Type | INTEGER_LABEL |
| Default value | 4 |
| Configuration class | PublishedInformation: |
| Origin | Elektrobit Automotive GmbH |

| Parameter Name | ArMinorVersion |
|---|---|
| Label | AUTOSAR Minor Version |
| Description | Minor version number of AUTOSAR specification on which the appropriate implementation is based on. |
| Multiplicity | 1..1 |
| Type | INTEGER_LABEL |
| Default value | 2 |
| Configuration class | PublishedInformation: |
| Origin | Elektrobit Automotive GmbH |

| Parameter Name | ArPatchVersion |
|---|---|
| Label | AUTOSAR Patch Version |
| Description | Patch level version number of AUTOSAR specification on which the appropriate implementation is based on. |

| | |
|---|---|
| **Multiplicity** | 1..1 |
| **Type** | INTEGER_LABEL |
| **Default value** | 1 |
| **Configuration class** | **PublishedInformation:** |
| **Origin** | Elektrobit Automotive GmbH |

| **Parameter Name** | **SwMajorVersion** |
|---|---|
| **Label** | Software Major Version |
| **Description** | Major version number of the vendor specific implementation of the module. |
| **Multiplicity** | 1..1 |
| **Type** | INTEGER_LABEL |
| **Default value** | 1 |
| **Configuration class** | **PublishedInformation:** |
| **Origin** | Elektrobit Automotive GmbH |

| **Parameter Name** | **SwMinorVersion** |
|---|---|
| **Label** | Software Minor Version |
| **Description** | Minor version number of the vendor specific implementation of the module. The numbering is vendor specific. |
| **Multiplicity** | 1..1 |
| **Type** | INTEGER_LABEL |
| **Default value** | 0 |
| **Configuration class** | **PublishedInformation:** |
| **Origin** | Elektrobit Automotive GmbH |

| **Parameter Name** | **SwPatchVersion** |
|---|---|
| **Label** | Software Patch Version |
| **Description** | Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific. |
| **Multiplicity** | 1..1 |
| **Type** | INTEGER_LABEL |
| **Default value** | 41 |
| **Configuration class** | **PublishedInformation:** |
| **Origin** | Elektrobit Automotive GmbH |

| Parameter Name | ModuleId |
|---|---|
| Label | Numeric Module ID |
| Description | Module ID of this module from Module List |
| Multiplicity | 1..1 |
| Type | INTEGER_LABEL |
| Default value | 175 |
| Configuration class | **PublishedInformation:** |
| Origin | Elektrobit Automotive GmbH |

| Parameter Name | VendorId |
|---|---|
| Label | Vendor ID |
| Description | Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list |
| Multiplicity | 1..1 |
| Type | INTEGER_LABEL |
| Default value | 1 |
| Configuration class | **PublishedInformation:** |
| Origin | Elektrobit Automotive GmbH |

| Parameter Name | Release |
|---|---|
| Label | Release Information |
| Multiplicity | 1..1 |
| Type | STRING_LABEL |
| Default value | |
| Configuration class | **PublishedInformation:** |
| Origin | Elektrobit Automotive GmbH |

#### 5.2.1.2. PublishedInformation

| Parameters included | |
|---|---|
| **Parameter name** | **Multiplicity** |
| PbcfgMSupport | 1..1 |

| Parameter Name | PbcfgMSupport |
|---|---|

| | |
|---|---|
| **Label** | PbcfgM support |
| **Description** | Specifies whether or not the ComXf can use the PbcfgM module for post-build support. |
| **Multiplicity** | 1..1 |
| **Type** | BOOLEAN |
| **Default value** | false |
| **Configuration class** | **PublishedInformation:** |
| **Origin** | Elektrobit Automotive GmbH |

### 5.2.1.3. XfrmGeneral

| **Parameters included** | |
|---|---|
| **Parameter name** | **Multiplicity** |
| XfrmBufferLengthType | 1..1 |
| XfrmDevErrorDetect | 1..1 |
| XfrmVersionInfoApi | 1..1 |

| **Parameter Name** | **XfrmBufferLengthType** | |
|---|---|---|
| **Description** | Specifies the data type of parameter BufferLength for transformer APIs. | |
| **Multiplicity** | 1..1 | |
| **Type** | ENUMERATION | |
| **Default value** | UINT16 | |
| **Range** | UINT16<br>UINT32 | |
| **Configuration class** | **VariantPreCompile:** | VariantPreCompile |
| **Origin** | Elektrobit Automotive GmbH | |

| **Parameter Name** | **XfrmDevErrorDetect** |
|---|---|
| **Description** | Switches the Development Error Detection and Notification ON or OFF.<br>► `TRUE`: Development Error Detection mechanism is enabled (switched on).<br>► `FALSE`: Development Error Detection mechanism is disabled (switched off). |
| **Multiplicity** | 1..1 |
| **Type** | BOOLEAN |

| | | |
|---|---|---|
| **Default value** | true | |
| **Configuration class** | **VariantPreCompile:** | VariantPreCompile |
| **Origin** | AUTOSAR_ECUC | |

| **Parameter Name** | **XfrmVersionInfoApi** | |
|---|---|---|
| **Description** | Activate/Deactivates the version information API. | |
| **Multiplicity** | 1..1 | |
| **Type** | BOOLEAN | |
| **Default value** | false | |
| **Configuration class** | **VariantPreCompile:** | VariantPreCompile |
| **Origin** | Elektrobit Automotive GmbH | |

### 5.2.1.4. XfrmImplementationMapping

| **Containers included** | | |
|---|---|---|
| **Container name** | **Multiplicity** | **Description** |
| XfrmVariableDataPrototypeInstanceRef | 1..1 | Instance reference to a VariableDataPrototype in case a dedicated transformer BswModuleEntry is required per VariableDataPrototype access. |
| XfrmDemEventParameterRefs | 1..1 | Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameters DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references. |
| XfrmSignal | 1..1 | Reference to the signal in the system description that transports the transformed data. |

| **Parameters included** | |
|---|---|
| **Parameter name** | **Multiplicity** |
| XfrmTransformerBswModuleEntryRef | 0..1 |
| XfrmInvTransformerBswModuleEntryRef | 0..1 |
| XfrmTransformationTechnologyRef | 1..1 |
| XfrmIsSafetyTransformer | 0..1 |

| Parameters included | |
|---|---|
| XfrmOsApplicationRef | 0..1 |

| Parameter Name | XfrmTransformerBswModuleEntryRef | |
|---|---|---|
| Description | Reference to the BswModuleEntry which implements the referenced transformer on the sending/calling side. | |
| Multiplicity | 0..1 | |
| Type | FOREIGN-REFERENCE | |
| Configuration class | **PreCompile:** | VariantPreCompile |
| Origin | Elektrobit Automotive GmbH | |

| Parameter Name | XfrmInvTransformerBswModuleEntryRef | |
|---|---|---|
| Description | Reference to the BswModuleEntry which implements the referenced inverse transformer on the receiving/called side. | |
| Multiplicity | 0..1 | |
| Type | FOREIGN-REFERENCE | |
| Configuration class | **PreCompile:** | VariantPreCompile |
| Origin | AUTOSAR_ECUC | |

| Parameter Name | XfrmTransformationTechnologyRef | |
|---|---|---|
| Description | Reference to the TransformationTechnology in the DataTransformation of the system description for which the implementation (BswModuleEntry) shall be mapped. | |
| Multiplicity | 1..1 | |
| Type | FOREIGN-REFERENCE | |
| Configuration class | **VariantPreCompile:** | VariantPreCompile |
| Origin | AUTOSAR_ECUC | |

| Parameter Name | XfrmIsSafetyTransformer | |
|---|---|---|
| Description | Specifies if the Transformer shall be considered as a safety Transformer. | |
| Multiplicity | 0..1 | |
| Type | BOOLEAN | |
| Default value | false | |
| Configuration class | **PreCompile:** | VariantPreCompile |
| Origin | Elektrobit Automotive GmbH | |

| Parameter Name | XfrmOsApplicationRef | |
|---|---|---|
| Description | Reference to an Os application to which the BSW belongs.<br>► `Enabled`: Maps global variables of transformer or inverted transformer function to dedicated memory partition.<br>► `Disabled`: No dedicated memory partition assigned. | |
| Multiplicity | 0..1 | |
| Type | REFERENCE | |
| Configuration class | **PreCompile:** | VariantPreCompile |
| Origin | Elektrobit Automotive GmbH | |

### 5.2.1.5. XfrmVariableDataPrototypeInstanceRef

| Parameters included | |
|---|---|
| **Parameter name** | **Multiplicity** |
| TARGET | 1..1 |
| CONTEXT | 2..2 |

| Parameter Name | TARGET | |
|---|---|---|
| Description | Target of the instance reference to a VariableDataPrototype.<br>The context of the instance reference is given by the list of Context References (CONTEXT) below. | |
| Multiplicity | 1..1 | |
| Type | REFERENCE | |
| Configuration class | **VariantPreCompile:** | VariantPreCompile |
| Origin | AUTOSAR_ECUC | |

| Parameter Name | CONTEXT |
|---|---|
| Description | List of ordered context references of the instance reference to:<br>1. SwComponentPrototype<br>2. PortPrototype<br>The target of the instance reference is given with the VariableDataPrototype reference (TARGET) above. |
| Multiplicity | 2..2 |

| | | |
|---|---|---|
| **Type** | REFERENCE | |
| **Range** | SW-COMPONENT-PROTOTYPE | |
| | PORT-PROTOTYPE* | |
| **Configuration class** | **PreCompile:** | VariantPreCompile |
| **Origin** | AUTOSAR_ECUC | |

### 5.2.1.6. XfrmDemEventParameterRefs

| **Parameters included** | |
|---|---|
| **Parameter name** | **Multiplicity** |
| XFRM_E_MALFORMED_MESSAGE | 0..1 |

| **Parameter Name** | **XFRM_E_MALFORMED_MESSAGE** | |
|---|---|---|
| **Description** | *The functionality related to this parameter is not supported by the current implementation.*<br>Reference to configured DEM event to report if malformed messages were received by the transformer. Reference to configured DEM event to report if malformed messages were received by the transformer. | |
| **Multiplicity** | 0..1 | |
| **Type** | SYMBOLIC-NAME-REFERENCE | |
| **Configuration class** | **VariantPreCompile:** | VariantPreCompile |
| **Origin** | AUTOSAR_ECUC | |

### 5.2.1.7. XfrmSignal

| **Containers included** | | |
|---|---|---|
| **Container name** | **Multiplicity** | **Description** |
| XfrmSignalChoice | 1..1 | Choice whether an ISignal or an ISignalGroup shall be referenced. |

### 5.2.1.8. XfrmSignalChoice

| **Containers included** | | |
|---|---|---|
| **Container name** | **Multiplicity** | **Description** |

| Containers included | | |
|---|---|---|
| XfrmISignalGroupRefChoice | 1..1 | Reference to the ISignalGroup in the system description that transports the transformed data. |
| XfrmISignalRefChoice | 1..1 | Reference to the ISignal in the system description that transports the transformed data. |

### 5.2.1.9. XfrmISignalGroupRefChoice

| Parameters included | |
|---|---|
| **Parameter name** | **Multiplicity** |
| XfrmISignalGroupRef | 1..1 |

| Parameter Name | XfrmISignalGroupRef | |
|---|---|---|
| Description | Reference to the ISignalGroup in the system description that transports the transformed data. | |
| Multiplicity | 1..1 | |
| Type | FOREIGN-REFERENCE | |
| Configuration class | **VariantPreCompile:** | VariantPreCompile |
| Origin | AUTOSAR_ECUC | |

### 5.2.1.10. XfrmISignalRefChoice

| Parameters included | |
|---|---|
| **Parameter name** | **Multiplicity** |
| XfrmISignalRef | 1..1 |

| Parameter Name | XfrmISignalRef | |
|---|---|---|
| Description | Reference to the ISignal in the system description that transports the transformed data. | |
| Multiplicity | 1..1 | |
| Type | FOREIGN-REFERENCE | |
| Configuration class | **VariantPreCompile:** | VariantPreCompile |
| Origin | AUTOSAR_ECUC | |

## 5.2.2. Application programming interface (API)

### 5.2.2.1. Type definitions

#### 5.2.2.1.1. ComXf_ConfigType

| | |
|---|---|
| **Purpose** | This is the type of the data structure containing the initialization data for the transformer. |
| **Type** | `uint8` |

### 5.2.2.2. Macro constants

#### 5.2.2.2.1. COMXF_E_INIT_FAILED

| | |
|---|---|
| **Purpose** | Error code if an invalid configuration set was selected. |
| **Value** | 0x02U |

#### 5.2.2.2.2. COMXF_E_PARAM

| | |
|---|---|
| **Purpose** | API Service called with wrong parameter. |
| **Value** | 0x03U |

#### 5.2.2.2.3. COMXF_E_PARAM_POINTER

| | |
|---|---|
| **Purpose** | API Service called with invalid pointer. |
| **Value** | 0x04U |

#### 5.2.2.2.4. COMXF_E_UNINIT

| | |
|---|---|
| **Purpose** | Error code if any other API service, except GetVersionInfo is called before the transformer module was initialized with Init or after a call to DeInit. |

| | |
|---|---|
| **Value** | 0x01U |

#### 5.2.2.2.5. COMXF_INSTANCE_ID

| | |
|---|---|
| **Purpose** | Id of instance of ComXf provided to Det_ReportError(). |
| **Value** | 0x00U |

#### 5.2.2.2.6. COMXF_SID_DEINIT

| | |
|---|---|
| **Purpose** | API Service ID for ComXf_DeInit(). |
| **Value** | 0x02U |

#### 5.2.2.2.7. COMXF_SID_GETVERSIONINFO

| | |
|---|---|
| **Purpose** | API Service ID for ComXf_GetVersioninfo(). |
| **Value** | 0x00U |

#### 5.2.2.2.8. COMXF_SID_INIT

| | |
|---|---|
| **Purpose** | API Service ID for ComXf_Init(). |
| **Value** | 0x01U |

#### 5.2.2.2.9. COMXF_SID_SR_INV_TRANSFORMER

| | |
|---|---|
| **Purpose** | API Service ID for ComXf_Inv_transformerId() of Sender/Receiver communication. |
| **Value** | 0x04U |

#### 5.2.2.2.10. COMXF_SID_SR_TRANSFORMER

| | |
|---|---|
| **Purpose** | API Service ID for ComXf_transformerId() of Sender/Receiver communication. |
| **Value** | 0x03U |

### 5.2.2.3. Functions

#### 5.2.2.3.1. ComXf_DeInit

| | |
|---|---|
| **Purpose** | Deinitialize the ComXf module. |
| **Synopsis** | `void` **`ComXf_DeInit`** `( void );` |
| **Service ID** | 0x02 |
| **Sync/Async** | Synchronous |
| **Reentrancy** | Non-Reentrant |
| **Description** | This function deinitializes the ComXf module. |

#### 5.2.2.3.2. ComXf_GetVersionInfo

| | | |
|---|---|---|
| **Purpose** | Get version information of the ComXf module. | |
| **Synopsis** | `void` **`ComXf_GetVersionInfo`** `( Std_VersionInfoType * VersionInfo );` | |
| **Service ID** | 0x00 | |
| **Sync/Async** | Synchronous | |
| **Reentrancy** | Reentrant | |
| **Parameters (out)** | `VersionInfo` | Pointer to where to store the version information of this module. |
| **Description** | This service returns the version information of this module. The version information includes:<br>► Module Id<br>► Vendor Id<br>► Vendor specific version numbers | |

#### 5.2.2.3.3. ComXf_Init

| | |
|---|---|
| **Purpose** | Initialize the ComXf module. |
| **Synopsis** | `void` **`ComXf_Init`** `( const ComXf_ConfigType * config );` |
| **Service ID** | 0x01 |
| **Sync/Async** | Synchronous |

| | | |
|---|---|---|
| **Reentrancy** | Non-Reentrant | |
| **Parameters (in)** | `config` | Points to the implementation specific structure |
| **Description** | This function initializes the ComXf module. | |

#### 5.2.2.3.4. ComXf_Inv_transformerId

| | | |
|---|---|---|
| **Purpose** | The COM deserializer interface for 'ComXf_Inv_<transformerId>'. | |
| **Synopsis** | `uint8 ComXf_Inv_transformerId ( const uint8 * buffer , uint16 bufferLength , < type > * dataElement );` | |
| **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 (out)** | `dataElement` | Data element which is the result of the transformation and contains the deserialized data element. |
| **Return Value** | Result of request to serialize the dataElement | |
| | `0x00` | (E_OK): Serialization successful |
| | `0x81` | (E_SER_GENERIC_ERROR): A generic error occurred |
| **Description** | This function deserializes a Sender/Receiver communication using the deserialization of COM Based Transformer. It takes the uint8 array containing the serialized data as input and outputs the original data element which will be passed to the Rte.<br><br>Preconditions: The COM module must be initialized. | |

#### 5.2.2.3.5. ComXf_transformerId

| | | |
|---|---|---|
| **Purpose** | The COM transformer interface for 'ComXf_<transformerId>'. | |
| **Synopsis** | `uint8 ComXf_transformerId ( uint8 * buffer , uint16 * bufferLength , const < type > * dataElement );` | |
| **Parameters (in)** | `dataElement` | Data element which shall be transformed. |
| **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** | Result of request to serialize the dataElement | |
| | `0x00` | (E_OK): Serialization successful |
| | `0x81` | (E_SER_GENERIC_ERROR): A generic error occurred |
| **Description** | This function transforms a Sender/Receiver communication using the serialization of COM Based Transformer. It takes the data element as input and outputs an uint8 ar- ray contain- ing the serialized data.<br><br>Preconditions: The COM module must be initialized. | |

## 5.2.3. Integration notes

### 5.2.3.1. Exclusive areas

Exclusive areas are not used by the `ComXf` module.

### 5.2.3.2. Production errors

Production errors are not reported by the `ComXf` module.

### 5.2.3.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The following table provides the list of sections that may be mapped for this module:

| **Memory section** |
|---|
| CODE |
| VAR_FAST_INIT_UNSPECIFIED |
| CONFIG_DATA_UNSPECIFIED |
| VAR_INIT_8 |
| VAR_NO_INIT_UNSPECIFIED |

| |
|---|
| CONST_32 |

### 5.2.3.4. Integration requirements

| | |
|---|---|
| **WARNING** | **Integration requirements list is not exhaustive**<br>The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user's guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product. |

#### 5.2.3.4.1. ComXf.EB.IntReq.RequiresCom01

| | |
|---|---|
| **Description** | The EB Com module shall be initialized before the invocation of ComXf_Init() and de-initialized only after ComXf_DeInit() returned. This behaviour only applies if a least one non safety related BSW entry is configured within the XfrmImplementationMapping of the ComXf module. |
| **Rationale** | The EB ComXf module requires the usage of Autocore Com module (reference to product description: ASCPD-288) if a least one non safety related BSW entry is configured within the XfrmImplementationMapping of the ComXf module. For these non safety related BSW entries the EB ComXf module uses the post-build configuration and library APIs of the EB Com module. It eases the implementation of the module. The configuration of the ComXf module and the Com module have to be consistent regarding the configuration of signal groups, which is ensured if the same configuration is used. |

#### 5.2.3.4.2. ComXf.EB.IntReq.InitRoutines

| | |
|---|---|
| **Description** | ComXf_Init() and ComXf_DeInit() must not be invoked from the context of a safe application. |
| **Rationale** | Different transformations might be mapped to different partitions. Any module-wide de-/initialization routine shall de-/initialize only non-safety relevant data. Partitions containing safetly relevant transformations shall provide their own individual de-/initialization routine (if needed). |

#### 5.2.3.4.3. ComXf.EB.IntReq.RequiresE2E01

| | |
|---|---|
| **Description** | If only safety related BSW entries are configured within the XfrmImplementationMapping of the ComXf module, no de-/initialization of the ComXf module is required. |

| | |
|---|---|
| **Rationale** | The EB ComXf module requires the usage of Autocore E2E library if a least one safety related BSW entry is configured within the XfrmImplementationMapping of the ComXf module. For these safety related BSW entries the EB ComXf module uses library macros of the EB E2E library and the system configuration itself. This means no usage of the post-build configuration and library APIs of the EB Com module applies. Nevertheless, the configuration of signal groups within the system configuration (for the ComXf module) and the Com module have to be consistent. |

#### 5.2.3.4.4. ComXf.EB.IntReq.EB_INTREQ_Com_0002

| | |
|---|---|
| **Description** | Restrictions to prevent race conditions in Com's Tx-path. The Com module exhibits several race conditions in its transmission path that can cause inconsistent and/or mutilated data to be transmitted. The transmission of an I-PDU can be triggered by a Tx-signal API if the I-PDU has a direct part (transmission mode is DIRECT or MIXED). The Tx-signal APIs are Com_SendSignal(), Com_SendDynSignal(), Com_SendSignalGroup(), and Com_SendSignalGroupArray(). The Tx-signal APIs have write access to the Com-internal I-PDU buffer. Note that (the internal implementations of) these APIs are also used in context of Com_MainFunctionRouteSignals(). Additionally the transmission of an I-PDU can be triggered in context of Com_MainFunctionTx(), Com_TriggerIPDUSend(), or Com_IpduGroupControl(), or Com_SwitchIpduTxMode. Triggering of a transmission in general requires the read access to the Com-internal I-PDU buffer by the Com lower layers. Depending on the implementation of a Tx-callout (ComIPduCallout and ComIPduTriggerTransmitCallout), it requires read and/or write access to the Com-internal I-PDU buffer. The callouts are invoked when a transmission is triggered. Depending on the underlying bus system, the API Com_TriggerTransmit() is invoked, which requires read access to the Com-internal I-PDU buffer. A race occurs when an ongoing transmission (access to the Com-internal I-PDU buffer by Com lower layer and Com callout) is interrupted by an invocation of a Tx-signal API. A race occurs when an ongoing transmission is interrupted by an API which triggers another transmission for the same I-PDU and a configured Com callout changes data. This behavior leads to the following cases:<br><br>- An I-PDU has a direct part. It also has a call to a Tx-signal API to a signal/signal group, in which one of the following transfer properties is interrupted by another Tx-signal API call of a signal of the very same I-PDU: TRIGGERED, TRIGGERED_ON_CHANGE, TRIGGERED_ON_CHANGE_WITHOUT_REPETITION, or TRIGGERED_WITHOUT_REPETITION.<br>- A call to a Tx-signal API for a signal/signal group that belongs to |

| | |
|---|---|
| | the I-PDU interrupts a call to one of the following APIs of the very same I-PDU: Com_TriggerIPDUSend(), Com_IpduGroupControl(), Com_SwitchIpduTxMode(), or Com_TriggerTransmit().<br>- A call to a Tx-signal API interrupts a call to Com_MainFunctionTx().<br>- A callout uses the data of the I-PDU for a calculation (e.g. to calculate a CRC) and a call to Tx-signal API interrupts the sending of the I-PDU.<br><br>With a call to Com_SendDynSignal() not only the content of an I-PDU may change, but also the length of the I-PDU. Work-around To prevent inconsistencies in the I-PDU, ensure the following:<br><br>- A call to a Tx-signal API that triggers a transmission does not interrupt a call to a Tx-signal API for signals which belong to the same I-PDU.<br>- A call to a Tx-signal API does not interrupt one of the following APIs: Com_TriggerIPDUSend(), Com_SwitchIpduTxMode(), or Com_TriggerTransmit().<br>- A call to a Tx-signal API does not interrupt Com_MainFunctionTx().<br>- Additionally, if a callout is configured that modifies I-PDU data: Ensure that the APIs: Com_TriggerIPDUSend()and Com_SwitchIpduTxMode() and Com_TriggerTransmit() and Com_MainFunctionTx() do not interrupt each other for the very same I-PDU. |
| **Rationale** | This issue could be avoided if you lock the PDU buffer or use expensive double buffers. However if you lock the PDU buffer while the callout function or the PduR_ComTransmit function is called, it leads to an undefined locking time. It is not acceptable to disable interrupts for too long. Therefore a usage restriction has been defined in the work-around section to avoid race conditions. |

#### 5.2.3.4.5. ComXf.EB.IntReq.EB_INTREQ_Com_0003

| | |
|---|---|
| **Description** | The access to the shadow buffer of a signal group is not protected. Therefore restrictions apply to the mutually possible preemptions. |

| | |
|---|---|
| | - On the Tx-side: A call to Com_UpdateShadowSignal() shall not get interrupted by Com_SendSignalGroup() for the signal group to which the group signal belongs to.<br>- On the Rx side: A call to Com_ReceiveShadowSignal() shall not get interrupted by Com_ReceiveSignalGroup() for the signal group to which the group signal belongs to. |
| **Rationale** | Restriction on allowed mutual preemptions. Work-around:<br>- Ensure that Com_SendSignalGroup() does not interrupt Com_UpdateShadowSignal() for the signal group to which the group signal belongs to.<br>- Ensure that Com_ReceiveSignalGroup() does not interrupt Com_ReceiveShadowSignal() for the signal group to which the group signal belongs to. |

#### 5.2.3.4.6. ComXf.EB.IntReq.EB_INTREQ_Com_0005

| | |
|---|---|
| **Description** | Limitation on Com signals/signal groups with update-bits. AUTOSAR COM SWS specifies that signals/signal groups with update-bits which have not been updated shall be discarded. However, if after an update of an I-PDU the value of a signal changes from e.g. x to y without the update bit is set, a call to Com_ReceiveSignal()/Com_ReceiveSignalGroup()-Com_ReceiveGroupSignal() returns the changed value (i.e. y) and not the last received value (i.e. x). Note: It is very unlikely that the receiver receives an updated value without the update-bit set. Because at sender side, the sender always sets the update-bit in case a new value is transmitted. The value of a signal/signal group only changes when the Com_SendSignal()/Com_SendSignalGroup() is invoked which sets the update-bit. An impact may only occur if the value on the sender is changed while the update-bit is not set. If this conditions occur this has no impact on the following use-cases:<br>- For applications (SWCs), at least if the EB-optimization DirectReadFromCom in Rte is not used. Since the Rte reads the value from the Com module only if it is notified by the Com module. This does not happen when the update-bit is not set. Also it writes the received value into a buffer and reads requests from the application and uses the value of the buffer.<br>- For applications which only use Com APIs when ComNotification is received. |

| | |
|---|---|
| | However, this conditions may have an impact on the following use-case: Applications, which directly use the Com APIs, usually get the correct value, since the value of a signal usually does not change without setting the update-bit. If you use the Com APIs without ComNotification, changed values may be read that have no update-bit set. The following work-around is only applicable in this case. Work-around for signals of type U/SINT8/16/32 Configure a filter (ComFilterAlgorithm) NEW_IS_WITHIN, with the parameters [ComFilterMin, ComFilterMax] = maximum possible value range. |
| **Rationale** | This limitation allows a more efficient implementation and for the application usually the behavior does not change. Requirements: - COM324 |