

Document Title	Specification of Module E2E Transformer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	650

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R19-11

Document Change History			
Date	Release	Changed by	Change Description
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> Incorporated usage of E2E_PxxForward methods to replicate detected E2E-Errors on outgoing messages Added Client-Server Communication support Updated Tracing from SRS_E2E to RS_E2E Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> Fix routine prototypes to correctly list optional parameters. correction applicable configuration parameter for datalength for profiles 2 and 22 Corrected reentrancy of E2EXf interfaces. Clarification of behavior and return value for DISABLE-END-TO-END-CHECK:TRUE.
2017-12-08	4.3.1	AUTOSAR Release Management	<ul style="list-style-type: none"> Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	<ul style="list-style-type: none"> Added support for Profiles P7, P11, P22 Various minor improvements
2015-07-31	4.2.2	AUTOSAR Release Management	<ul style="list-style-type: none"> Various minor fixes

Document Change History

Date	Release	Changed by	Change Description
2014-10-31	4.2.1	AUTOSAR Release Management	<ul style="list-style-type: none">Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1	Introduction and functional overview	5
2	Acronyms and abbreviations	7
3	Related documentation.....	8
3.1	Input documents	8
3.2	Related standards and norms	8
3.3	Related specification	9
4	Constraints and assumptions	10
4.1	Limitations	10
4.2	Applicability to car domains	10
5	Dependencies to other modules.....	11
5.1	Supported configuration variants.....	11
6	Requirements traceability	12
7	Functional specification	13
7.1	Supported RTE functions	13
7.2	Naming for functions and data to be protected by E2E	13
7.3	Generated structure types	15
7.4	Static initialization	16
7.5	Runtime initialization by E2EXf_Init() function	19
7.6	Normal operation.....	20
7.7	Error classification	31
8	API specification.....	34
8.1	Imported types.....	34
8.2	Type definitions	35
8.3	Function definitions.....	36
8.4	Call-back notifications.....	40
8.5	Scheduled functions	40
8.6	Expected Interfaces.....	40
9	Sequence diagrams	45
9.1	Protect – E2EXf_<transformerId>	45
9.2	Check – E2EXf_Inv_<transformerId>	46
10	Configuration specification.....	47
11	Not applicable requirements	48

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module E2E Transformer.

E2E Transformer belongs to the class “Safety”, according to SRS Transformer General.

The E2E transformer ensures a correct communication of I-signals through QM communication stack. The communication stack is considered as “black channel” communication.

There is a one-to-one relationship between a data element and I-signal, in the sense that there is no data splitting/merging (i.e. one I-signal is NOT made of several data elements, one data element is not made of several I-signals):

1. On the sender side, one data element maps to exactly one-to-one to an I-signal.
2. On the receiver side, one-or-more data elements represent the entire received I-signal (i.e. receiver fan-out).

There is a fan-out I-signal to one-or-more data elements on receiver side. The following scenarios are supported:

3. On sender side, one data element serialized to one I-signal, and protected with one E2E-protection
4. One or more I-signals placed in one I-PDU, where each I-signal has a different E2E protection. Some I-signals may have no E2E protection at all.
5. On receiver side, one I-signal checked:
 - a. Once: resulting with one data element (i.e. no fan-out)
 - b. Several times, with the same settings, but by independent functions (e.g. by ASIL-independent receiver SW-Cs),
 - c. Several times: with partially different settings (e.g. same DataID, but different counter tolerances), by different functions, resulting with separate data elements each having possibly different E2E-check result,
 - d. Several times: with and without E2E-check enabled (e.g. if one receiver is safety-related, and another one is QM and it does not need the results of E2E check).

The E2E Transformer is responsible for the invocation of the E2E Library based on the configuration of specific data element (I-signal).

The E2E Transformer instantiates the E2E configuration and E2E state data structures, based on its configuration. All E2E profiles may be used to protect data.

The E2E Transformer encapsulates the complexity of configuring and handling of the E2E and it offers a standard Transformer interface. Thanks to this, the caller of E2E Transformer does not need to know the E2E internals.

The E2E Transformer is invoked by RTE, and the RTE invocation is a result of invocation of RTE API (read, write, send, receive) by software components.

2 Acronyms and abbreviations

See AUTOSAR glossary.

3 Related documentation

3.1 Input documents

- [1] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf
- [2] AUTOSAR General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf
- [3] AUTOSAR General Specification for Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf
- [4] AUTOSAR Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_SWS_E2ELibrary.pdf
- [5] AUTOSAR Specification of RTE
AUTOSAR_SWS_RTE.pdf
- [6] AUTOSAR Requirements on E2E
AUTOSAR_RS_E2E.pdf
- [7] System Template
AUTOSAR_TPS_SystemTemplate.pdf
- [8] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf
- [9] General Specification of Transformers
AUTOSAR_ASWS_TransformerGeneral.pdf
- [10] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf
- [11] SoftwareComponent Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

3.2 Related standards and norms

- [12] ISO 26262:2011
www.iso.org

3.3 Related specification

AUTOSAR provides a General Specification on Transformers (ASWS Transformer General)[9], which is also valid for E2E Transformer.

Thus, the specification ASWS Transformer General[9] shall be considered as additional and required specification for E2E Transformer.

4 Constraints and assumptions

4.1 Limitations

The current solution for E2E communication protection is based on some use case constraints. Possibly, they will be reduced or they will be removed in future document versions:

1. For sender/receiver communication only cyclic communication of/between Software Components is considered (no event-based),
2. Only inter-ECU communication is considered (communication that is exchanged over COM stack),
3. Only non-blocking characteristics of queued sender-receiver communication is considered (blocking characteristic for queued communication is not supported)
4. NoNewData checks are only supported for polling-based Client-Server Communication and Sender-Receiver Communication.
5. E2E Profiles 1 and 2 do not support Client-Server Communication at all.

Please note that these constraints do not necessarily limit the use of the E2E Transformer to use cases meeting these constraints.

Further, the following limitations are known:

1. Error reporting to DEM not yet specified.

Cyclic invocation is needed because the E2E_Check is responsible (among others) to detect losses and delays, so in case of loss or delay, the E2E_Check is not invoked, resulting with potentially lost error detection. The cyclic invocation of E2E Transformer is ensured by cyclic invocation of corresponding RTE functions by Software Components.

4.2 Applicability to car domains

The E2E Transformer is applicable for safety-related communication.

5 Dependencies to other modules

The E2E Transformer depends on E2E Library. E2E Library provides data types and virtual (i.e. stateless) functions. E2E Transformer executes the E2E Library routines passing the configuration and state as function parameters.

5.1 Supported configuration variants

This document – SWS E2E Transformer – specifies the details of two configuration variants:

1. Link-time
2. Post-build-selectable.

Both are quite similar and use same kind of data structures. In link-time, structures are instantiated only once, while in post-build-selectable they can be instantiated more than once (with different values). Moreover, the configuration structures reside in different files (with suffix LCfg vs. PBCfg, see below).

There are currently no explicit Pre-compile time configuration settings apart from the settings specified in BSW General [3] and ASWS TransformerGeneral[9].

6 Requirements traceability

Requirement	Description	Satisfied by
RS_E2E_08528	E2E protocol shall provide different E2E profiles	SWS_E2EXf_00158, SWS_E2EXf_00159
RS_E2E_08538	An E2E transformer shall be provided	SWS_E2EXf_00009, SWS_E2EXf_00018, SWS_E2EXf_00021, SWS_E2EXf_00024, SWS_E2EXf_00027, SWS_E2EXf_00029, SWS_E2EXf_00032, SWS_E2EXf_00035, SWS_E2EXf_00037, SWS_E2EXf_00087, SWS_E2EXf_00089, SWS_E2EXf_00096, SWS_E2EXf_00102, SWS_E2EXf_00104, SWS_E2EXf_00106, SWS_E2EXf_00108, SWS_E2EXf_00111, SWS_E2EXf_00113, SWS_E2EXf_00115, SWS_E2EXf_00118, SWS_E2EXf_00120, SWS_E2EXf_00123, SWS_E2EXf_00125, SWS_E2EXf_00130, SWS_E2EXf_00133, SWS_E2EXf_00137, SWS_E2EXf_00139, SWS_E2EXf_00141, SWS_E2EXf_00144, SWS_E2EXf_00146, SWS_E2EXf_00149, SWS_E2EXf_00151, SWS_E2EXf_00153, SWS_E2EXf_00155, SWS_E2EXf_00162, SWS_E2EXf_00164, SWS_E2EXf_00166, SWS_E2EXf_00167, SWS_E2EXf_00011, SWS_E2EXf_00020, SWS_E2EXf_00023, SWS_E2EXf_00025, SWS_E2EXf_00028, SWS_E2EXf_00030, SWS_E2EXf_00034, SWS_E2EXf_00036, SWS_E2EXf_00048, SWS_E2EXf_00088, SWS_E2EXf_00090, SWS_E2EXf_00097, SWS_E2EXf_00103, SWS_E2EXf_00105, SWS_E2EXf_00107, SWS_E2EXf_00109, SWS_E2EXf_00112, SWS_E2EXf_00114, SWS_E2EXf_00116, SWS_E2EXf_00119, SWS_E2EXf_00122, SWS_E2EXf_00124, SWS_E2EXf_00126, SWS_E2EXf_00132, SWS_E2EXf_00134, SWS_E2EXf_00138, SWS_E2EXf_00140, SWS_E2EXf_00142, SWS_E2EXf_00145, SWS_E2EXf_00148, SWS_E2EXf_00150, SWS_E2EXf_00152, SWS_E2EXf_00154, SWS_E2EXf_00161, SWS_E2EXf_00163, SWS_E2EXf_00165,
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_E2EXf_00156

7 Functional specification

E2E transformer is responsible for protecting safety-related data elements. It is invoked by RTE. On the sender side, E2E transformer E2E-protects the data. On the receiver side, E2E transformer E2E-checks the data, providing the result of the E2E-checks through RTE to SW-C.

If a receiving SWC doesn't read the transformer return codes, it is fully transparent to the communicating SWCs whether the data are E2E protected on the bus or not.

All algorithms are provided by E2E Library (protect, check, state machine). E2E transformer invokes E2 Library providing the configuration and state.

E2E transformer is generated to a high extent, where both configuration data structures and functions are generated.

The E2E Transformer has no module specific ECU configuration because its whole configuration is based on the E2ETransformationDescription, the E2ETransformationIProps and the E2ETransformationComSpecProps. Thus the generic ECU configuration of the ASWS Transformer General[9] is sufficient.

The configuration input can be found in TPS SystemTemplate ([7]) and TPS SoftwareComponentTemplate ([11]).

[SWS_E2EXf_00161] The E2E transformer defined in this document shall be used as a transformer if

1. the attribute protocol of the TransformationTechnology is set to E2E
2. and the attribute version of the TransformationTechnology is set to 1.0.0
3. and the attribute transformerClass of the TransformationTechnology is set to safety] (RS_E2E_08538)

7.1 Supported RTE functions

Currently, the following inter-ECU communication functions are supported:

1. Rte_Write/Rte_Read
2. Rte_IWrite/Rte_IRead
3. Rte_Send/Rte_Receive
4. Rte_Call/Rte_Result

7.2 Naming for functions and data to be protected by E2E

E2E Transformer functions and structures get the suffix <transformerId>, defined as follows.

The pattern <transformerId> is defined in [SWS_Xfrm_00062] of ASWS Transformer General [9] and defines a unique ID for each transformer function.

This name pattern is also used in the names of the E2E transformer's C-APIs and therefore used in the BswModuleEntrys which represent the C-APIs.

This configuration builds the three stages of transformer configuration:

- **EndToEndTransformationDescription**
defines the E2E configuration profiles, valid for several ISignals
- **EndToEndTransformationISignalProps**
defines the configuration options valid for a specific referenced ISignal
- **EndToEndTransformationComSpecProps**
defines the override configuration options valid for the port to which the ReceiverComSpec belongs

It is possible that there are several software components receiving independently data elements that are created (deserialized) from the same I-PDU. The following cases are possible:

1. Some software components have adjusted/special configuration values, related to the tolerances of the E2E state machine, e.g. bigger tolerances. For this, attributes of EndToEndTransformationComSpecProps is used.
2. Some QM software components may not need to E2E-check the data, so the E2E check can be skipped. For this, EndToEndTransformationComSpecProps.disableEndToEndCheck is used.

[SWS_E2EXf_00134] The configuration options in EndToEndTransformationComSpecProps shall have precedence over the options in EndToEndTransformationDescription and EndToEndTransformationISignalProps.] (RS_E2E_08538)

That means:

Configuration options in EndToEndTransformationComSpecProps override the configuration options in EndToEndTransformationDescription and EndToEndTransformationISignalProps.

[SWS_E2EXf_00154] If configuration option EndToEndTransformationComSpecProps.disableEndToEndCheck is set for a given <transformerId>, then E2E Transformer shall skip the invocation of the E2E Library – it shall only perform buffer processing (e.g. copying from inputBuffer to buffer). Return value shall be E_OK.] (RS_E2E_08538)

To support multiple post-build-selectable variants, each configuration has a variant identifier.

[SWS_E2EXf_00090] In case of post-build-selectable configuration, the variants shall be named according to the configuration attribute PredefinedVariant.shortName. This means:
<v> = PredefinedVariant.shortName.] (RS_E2E_08538)

[SWS_E2EXf_00089] In case of link-time configuration, there is just one variant, this means:

<v>= empty (NULL string).] (RS_E2E_08538)

Note that all variants that are based on the same TransformationTechnology use the same E2E profile (e.g. P04).

This also means that all transformers with the same <transformerId> use the same E2E profile.

All variants have the same E2E-protected data elements.

The functions and state-structures are independent on variants <v> - they depend only on the specific instance of the E2E transformer and therefore on the <transformerId>, whereas config-structures depends on instance (<transformerId>) and configuration variant(<v>).

7.3 Generated structure types

Based on the E2E Transformer configuration (described in SystemTemplate [7], SoftwareComponentTemplate [11]) and the generated ECU configuration (described in ASWS Transformer General [9]), the corresponding C structures are generated as described below.

7.3.1 Overall config and state of E2E Transformer

[SWS_E2EXf_00011] The E2E Transformer shall generate the following data structure, to store the configuration of E2E Transformer module:

E2EXf_ConfigStruct_<v>(of type E2EXf_ConfigType)] (RS_E2E_08538)

[SWS_E2EXf_00125] The E2E Transformer shall derive the required number of independent state data resources of types E2E_PXXProtectStateType, E2E_PXXCheckStateType, and E2E_SMCheckStateType to perform E2E Protection within the E2E Transformer module from the number of E2E-protected data uniquely identified with <transformerId>, protected by profile XX.] (RS_E2E_08538)

7.3.2 Config and state of each E2E-protected data

[SWS_E2EXf_00126] The E2E Transformer shall derive the required number of independent statically initialized configuration objects of types E2E_PXXConfigType and E2E_SMConfigType to perform E2E Protection within the E2E Transformer, from:

1. the number of E2E-protected data uniquely identified with <transformerId>, protected by profile XX, and
2. the number of configuration variants (post-build selectable only).] (RS_E2E_08538)

7.4 Static initialization

7.4.1 Static initialization of config

Configuration is statically initialized based on the following metamodel classes:

1. EndToEndTransformationDescription: definition of E2E variants
2. EndToEndTransformationISignalProps: definition of a specific protection for a given ISignal (e.g. length, DataID)
3. EndToEndTransformationComSpecProps: override of some settings defining the check tolerances, with respect to E2E variants.

[SWS_E2Exf_00048] The generated configuration object of type E2E_P01ConfigType shall be initialized according to the following:

- DataID = EndToEndTransformationISignalProps.dataID
- DataLength = EndToEndTransformationISignalProps.dataLength
- CounterOffset = EndToEndTransformationDescription.counterOffset
- CRCOffset = EndToEndTransformationDescription.crcOffset
- DataIDNibbleOffset = EndToEndTransformationDescription.dataIDNibbleOffset
- DataIDMode shall be set to
 - E2E_P01_DATAID_BOTH if
EndToEndTransformationDescription.dataIDMode == all16Bit
 - E2E_P01_DATAID_ALT if
EndToEndTransformationDescription.dataIDMode == alternating8Bit
 - E2E_P01_DATAID_LOW if
EndToEndTransformationDescription.dataIDMode == lower8Bit
 - E2E_P01_DATAID_NIBBLE if
EndToEndTransformationDescription.dataIDMode == nibble
- MaxDeltaCounterInit =
EndToEndTransformationComSpecProps.maxDeltaCounter-1 or
EndToEndTransformationDescription.maxDeltaCounter-1
- MaxNoNewOrRepeatedData =
EndToEndTransformationComSpecProps.maxNoNewOrRepeatedData or
EndToEndTransformationDescription.maxNoNewOrRepeatedData
- SyncCounterInit = EndToEndTransformationComSpecProps.syncCounterInit
or EndToEndTransformationDescription.syncCounterInit.] (RS_E2E_08538)

[SWS_E2Exf_00118] The generated configuration object of type E2E_P02ConfigType shall be initialized according to the following:

- DataIDList= EndToEndTransformationISignalProps.dataID (array)
- DataLength = EndToEndTransformationISignalProps.dataLength
- Offset = EndToEndTransformationDescription.offset

- MaxDeltaCounterInit = EndToEndTransformationComSpecProps.maxDeltaCounter-1 or EndToEndTransformationDescription.maxDeltaCounter-1
- MaxNoNewOrRepeatedData = EndToEndTransformationComSpecProps.maxNoNewOrRepeatedData or EndToEndTransformationDescription.maxNoNewOrRepeatedData
- SyncCounterInit = EndToEndTransformationComSpecProps.syncCounterInit or EndToEndTransformationDescription.syncCounterInit.] (RS_E2E_08538)

[SWS_E2EXf_00087] The generated configuration object of type E2E_P04ConfigType shall be initialized according to the following (one-to-one mapping):

- DataID = EndToEndTransformationISignalProps.dataID
- MinDataLength = EndToEndTransformationISignalProps.minDataLength
- MaxDataLength = EndToEndTransformationISignalProps.maxDataLength
- Offset = EndToEndTransformationDescription.offset
- MaxDeltaCounter = EndToEndTransformationComSpecProps.maxDeltaCounter or EndToEndTransformationDescription.maxDeltaCounter] (RS_E2E_08538)

[SWS_E2EXf_00119] The generated configuration object of type E2E_P05ConfigType shall be initialized according to the following (one-to-one mapping):

- DataID = EndToEndTransformationISignalProps.dataID
- DataLength = EndToEndTransformationISignalProps.dataLength
- Offset = EndToEndTransformationDescription.offset
- MaxDeltaCounter = EndToEndTransformationComSpecProps.maxDeltaCounter or EndToEndTransformationDescription.maxDeltaCounter] (RS_E2E_08538)

[SWS_E2EXf_00120] The generated configuration object of type E2E_P06ConfigType shall be initialized according to the following (one-to-one mapping):

- DataID = EndToEndTransformationISignalProps.dataID
- MinDataLength = EndToEndTransformationISignalProps.minDataLength
- MaxDataLength = EndToEndTransformationISignalProps.maxDataLength
- Offset = EndToEndTransformationDescription.offset
- MaxDeltaCounter = EndToEndTransformationComSpecProps.maxDeltaCounter or EndToEndTransformationDescription.maxDeltaCounter] (RS_E2E_08538)

[SWS_E2EXf_00163] The generated configuration object of type E2E_P07ConfigType shall be initialized according to the following (one-to-one mapping):

- DataID = EndToEndTransformationISignalProps.dataID

- MinDataLength = EndToEndTransformationISignalProps.minDataLength
- MaxDataLength = EndToEndTransformationISignalProps.maxDataLength
- Offset = EndToEndTransformationDescription.offset
- MaxDeltaCounter =
EndToEndTransformationComSpecProps.maxDeltaCounter or
EndToEndTransformationDescription.maxDeltaCounter] (RS_E2E_08538)

[SWS_E2EXf_00162] The generated configuration object of type E2E_P11ConfigType shall be initialized according to the following (one-to-one mapping):

- DataID = EndToEndTransformationISignalProps.dataID
- DataLength = EndToEndTransformationISignalProps.dataLength
- CounterOffset = EndToEndTransformationDescription.counterOffset
- CRCOffset = EndToEndTransformationDescription.crcOffset
- DataIDNibbleOffset = EndToEndTransformationDescription.dataIDNibbleOffset
- DataIDMode shall be set to
 - E2E_P11_DATAID_BOTH if
EndToEndTransformationDescription.dataIDMode == all16Bit
 - E2E_P11_DATAID_NIBBLE if
EndToEndTransformationDescription.dataIDMode == nibble
- MaxDeltaCounter =
EndToEndTransformationComSpecProps.maxDeltaCounter or
EndToEndTransformationDescription.maxDeltaCounter] (RS_E2E_08538)

[SWS_E2EXf_00164] The generated configuration object of type E2E_P22ConfigType shall be initialized according to the following:

- DataIDList = EndToEndTransformationISignalProps.dataID (array)
- DataLength = EndToEndTransformationISignalProps.dataLength
- Offset = EndToEndTransformationDescription.offset
- MaxDeltaCounter =
EndToEndTransformationComSpecProps.maxDeltaCounter or
EndToEndTransformationDescription.maxDeltaCounter

] (RS_E2E_08538)

[SWS_E2EXf_00088] The generated config structure of type E2E_SMConfigType, shall be initialized according to the following (one-to-one mapping):

- WindowSize = EndToEndTransformationComSpecProps.windowSize or
EndToEndTransformationDescription.windowSize
- MinOkStateInit = EndToEndTransformationComSpecProps.minOkStateInit or
EndToEndTransformationDescription.minOkStateInit
- MaxErrorStateInit =
EndToEndTransformationComSpecProps.maxErrorStateInit or
EndToEndTransformationDescription.maxErrorStateInit

- MinOkStateValid = EndToEndTransformationComSpecProps.minOkStateValid or EndToEndTransformationDescription.minOkStateValid
- MaxErrorStateValid = EndToEndTransformationComSpecProps.maxErrorStateValid or EndToEndTransformationDescription.maxErrorStateValid
- MinOkStateInvalid = EndToEndTransformationComSpecProps.minOkStateInvalid or EndToEndTransformationDescription.minOkStateInvalid
- MaxErrorStateInvalid = EndToEndTransformationComSpecProps.maxErrorStateInvalid or EndToEndTransformationDescription.maxErrorStateInvalid] (RS_E2E_08538)

[SWS_E2EXf_00096] The configuration object E2EXf_ConfigStruct_<v> (see SWS_E2EXf_00011) shall be initialized to contain or to reference the config structures that were instantiated in above requirements of this section.] (RS_E2E_08538)

[SWS_E2EXf_00097] In link-time variant, the E2EXf_Config pointer shall be initialized to reference E2EXf_ConfigStruct_0.] (RS_E2E_08538)

For post-build-selectable variant, the above code snippets are similar, but the initialization is done for every variant. Moreover, E2EXf_Config is set to NULL.

7.4.2 Static Initialization of state

Contrary to config structures, state structures do not depend on variants (<v>).

[SWS_E2EXf_00023] In all E2E Transformer variants, the generated state objects may be left uninitialized (i.e. without providing explicit initialization values).] (RS_E2E_08538)

7.5 Runtime initialization by E2EXf_Init() function

7.5.1 Runtime selection of configuration (post-build variant only)

[SWS_E2EXf_00024] In post-build-selectable variant, E2EXf_Init() shall check that Config pointer (received as function parameter) points to one of the configuration variants E2EXf_ConfigStruct_<v>. If it is equal, then E2EXf_Init() shall select the

passed configuration variant, and it shall set the module initialization state to TRUE according to SWS_E2EXf_00130.] (RS_E2E_08538)

7.5.2 Runtime initialization of State

[SWS_E2EXf_00021] The E2EXf_Init() function shall initialize all external state data resources managed by E2E transformer (see SWS_E2EXf_00125) as follows:

- Initialization of state data resources of type E2E_PXXProtectStateType by calling corresponding E2E_PXXProtectInit() methods,
- Initialization of state data resources of type E2E_PXXCheckStateType by calling corresponding E2E_PXXCheckInit() methods,
- Initialization of state data resources of type E2E_SMCheckStateType by calling corresponding E2E_SMCheckInit() methods.] (RS_E2E_08538)

[SWS_E2EXf_00158] The E2EXf_Init() function shall initialize all internal state data resources of E2E transformer.] (RS_E2E_08528)

[SWS_E2EXf_00159] In case of post-build configuration, E2EXf_Init() function shall store the information about the selected configuration.] (RS_E2E_08528)

[SWS_E2EXf_00130] The E2E Transformer shall maintain a boolean information (Initialization state) that is only set to TRUE, if the module has been successfully initialized via a call to E2EXf_Init(). Otherwise, it is set to FALSE.] (RS_E2E_08538)

[SWS_E2EXf_00132] In case of deinitialization (invocation of E2EXf_DeInit()), the module initialization state shall be set to FALSE.] (RS_E2E_08538)

7.6 Normal operation

[SWS_E2EXf_00133] If the E2E Transformer has not been correctly initialized (which means that E2EXf_Init() was not successfully called before), then all generated E2E Transformer APIs shall immediately return E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

7.6.1 In-place processing and out-of-place processing

E2E Transformer functions work using in-place processing or out-of-place processing. This is configured by binary setting BufferProperties.inPlace.

In-place means that one buffer is used by a transformer both as input and as output. In-place processing has a performance advantage (less copying, less buffers). Out-of-place means that there is one input buffer and a separate output buffer.

7.6.2 E2EXf_<transformerId>(protect-function)

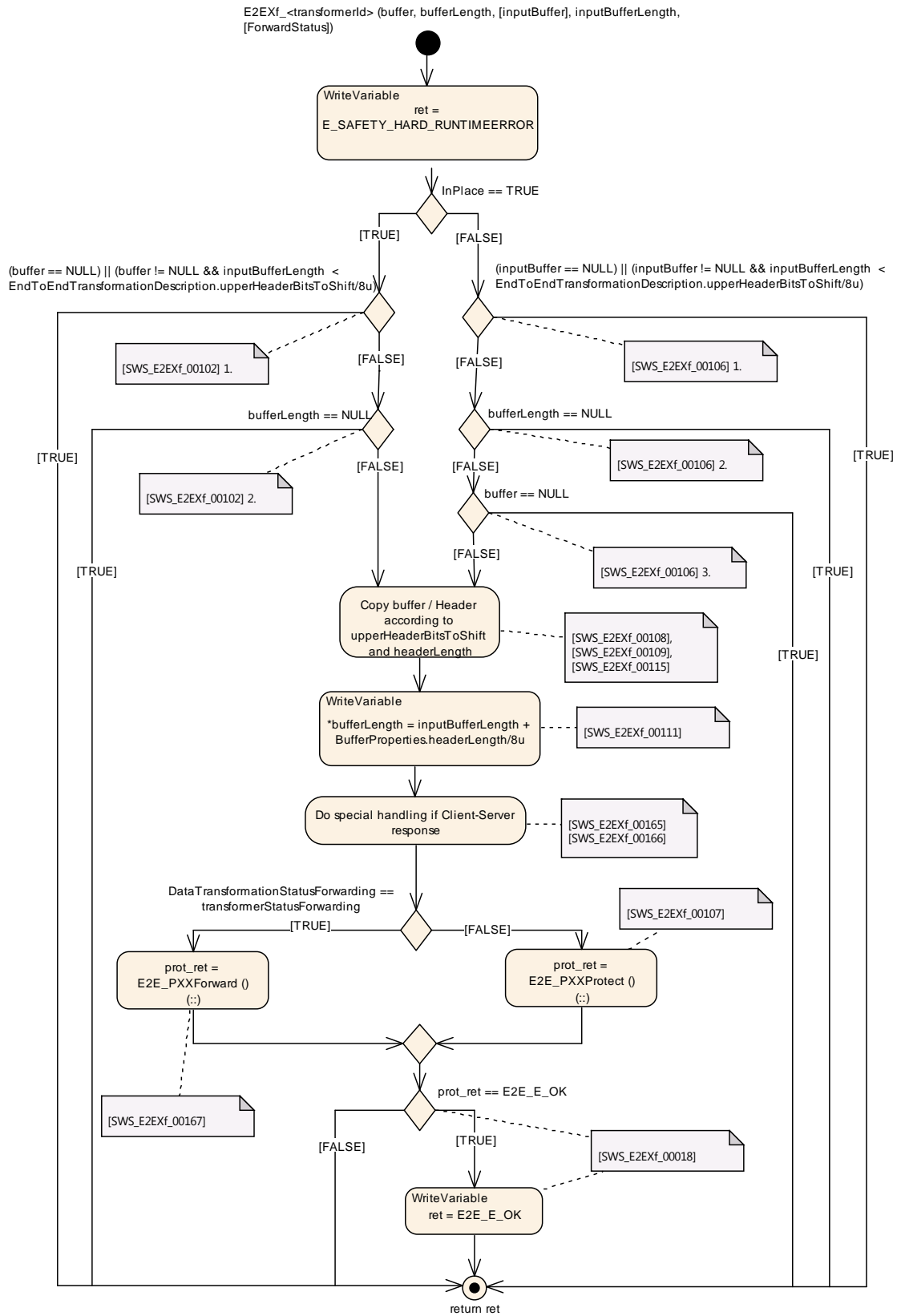


Figure 7-1: E2EXf_<transformerId>function overview

Figure above provides an activity diagram of the functionality provided by the API function E2EXf_<transformerId>.

[SWS_E2EXf_00020] The function E2EXf_<transformerId> shall be generated for each E2E-protected data element.] (RS_E2E_08538)

[SWS_E2EXf_00102] In-place E2EXf_<transformerId> shall perform the following two precondition checks, without continuing further processing:

1. (buffer == NULL)
||
(buffer != NULL && inputBufferLength < EndToEndTransformationDescription.upperHeaderBitsToShift/8u)
2. bufferLength == NULL.

If any of above conditions is TRUE, then the function shall return E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

[SWS_E2EXf_00106] Out-of-place E2EXf_<transformerId> shall perform the following three precondition checks, without continuing further processing:

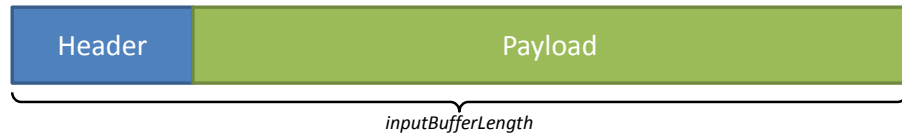
1. (inputBuffer == NULL)
||
(inputBuffer != NULL && inputBufferLength < EndToEndTransformationDescription.upperHeaderBitsToShift/8u)
2. bufferLength == NULL
3. buffer == NULL

If any of above conditions is TRUE, then the function shall return E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

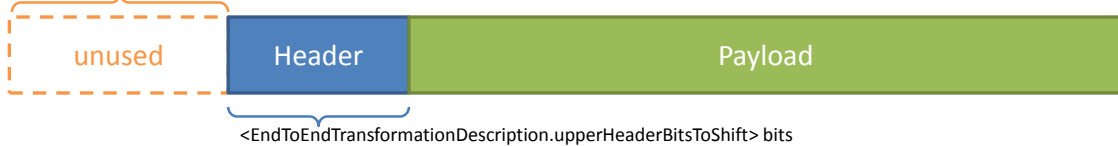
Note that the function E2EXf_<transformerId> can be realized by a plain function or a macro (implementation-specific). The functions E2EXf_<transformerId> may call some internal common functions.

[SWS_E2EXf_00108] If (EndToEndTransformationDescription.upperHeaderBitsToShift > 0), in-place E2EXf_<transformerId> shall copy the amount upper HeaderBitsToShiftbits, in parameter buffer, with starting offset of BufferProperties.headerLength, in direction left by “distance” of BufferProperties.headerLength.] (RS_E2E_08538)

Previous transformer's output:



E2E-Transformer gets a pointer to a buffer with $\langle \text{BufferProperties.headerLength} \rangle$ leading bits:
 $\langle \text{BufferProperties.headerLength} \rangle$ bits



E2E-Transformer copies header to front [SWS_E2EXf_00108]:



Figure 7-2: Buffer handling of E2EXf_<transformerId>

Figure 7-2 illustrates the buffer handling done by API function E2EXf_<transformerId> for In-Place.

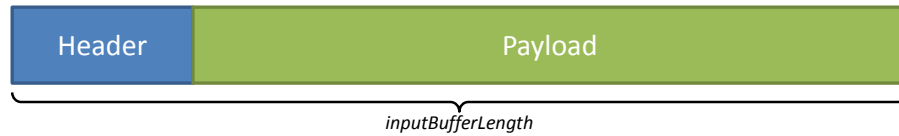
[SWS_E2EXf_00109] If

(EndToEndTransformationDescription.upperHeaderBitsToShift > 0), out-of-place E2EXf_<transformerId> shall copy the first upper HeaderBitsToShiftbits from inputBuffer to buffer, and then copy the remaining part of inputBuffer (i.e. starting with offset upperHeaderBitsToShift) to parameter buffer starting with the destination offset of (upperHeaderBitsToShift+ BufferProperties.headerLength).] (RS_E2E_08538)

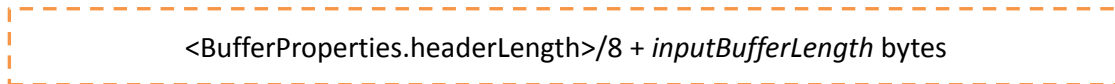
[SWS_E2EXf_00115] If

(EndToEndTransformationDescription.upperHeaderBitsToShift == 0), out-of-place E2EXf_<transformerId> shall copy inputBuffer to buffer starting with the destination offset of BufferProperties.headerLength.] (RS_E2E_08538)

Previous transformer's output:



E2E-Transformer gets a pointer to an empty buffer of size $\langle \text{BufferProperties.headerLength} \rangle / 8 + \text{inputBufferLength}$ bytes:



E2E-Transformer copies header to front, payload to back [SWS_E2EXf_00109] :

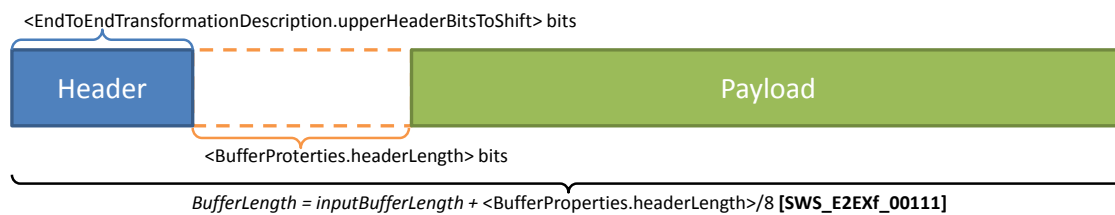


Figure 7-3: E2EXf_<transformerId> header shift Out-of-place

Figure 7-3 illustrates the buffer handling done by API function E2EXf_<transformerId> for Out-of-place.

[SWS_E2EXf_00111] E2EXf_<transformerId> shall set `*bufferLength = inputBufferLength + BufferProperties.headerLength/8.` (RS_E2E_08538)

[SWS_E2EXf_00139] For PXX = 01 or 02, the function E2EXf_<transformerId>() shall perform a check of the `*bufferLength` (after the computation of `*bufferLength`): If (`*bufferLength != config->DataLength/8`), then the function shall return immediately `E_SAFETY_HARD_RUNTIMEERROR`, i.e. without calling an E2E Library function.] (RS_E2E_08538)

In case Some/IP based transformer is used, the E2E header of profile 1, 2, 11 and 22 are extended to 2 bytes. The extension is done by filling up unused nibble (4 bits) with 0xF. E2E header, when used with Some/IP, is always a consecutive block of bits, so the empty position is always the same: it is the high nibble of the byte after the CRC byte:

[SWS_E2EXf_00155] If (((PXX = 01 &&dataIDMode != nibble) || (PXX == 02) || (PXX = 11 &&dataIDMode != nibble) || (PXX == 22)) && BufferProperties.headerLength == 16 [bits]), the function E2EXf_<transformerId>() shall, before calling E2E_PXXProtect() or E2E_PXXForward(), set 0xF in buffer at

the bit offset (EndToEndTransformationDescription.crcOffset+12 for profiles P01 and P11 and EndToEndTransformationDescription.offset+12 for profiles P02 and P22).] (RS_E2E_08538)

[SWS_E2EXf_00165] If E2E-Transformer is used in a Client-Server Communication and Profile is P02 or P22 the sequence counter used for protecting the response shall be set to receivedCounter-1.] (RS_E2E_08538)

[SWS_E2EXf_00166] If E2E-Transformer is used in a Client-Server Communication and Profile is P01, P04, P05, P06, P07 or P11 the sequence counter for protecting the response shall be set to receivedCounter.] (RS_E2E_08538)

Implementation Hint: Access E2E_PXXCheckStateType* State, of the corresponding E2EXf_Inv inside the E2EXf call to get the last received counter for this Client-Server-Communication, this only works under the assumption connected Request-Response of a Client-Server-Call are handled synchronously.

[SWS_E2EXf_00107] If DataTransformationStatusForwarding is set to noTransformerStatusForwarding, the function E2EXf_<transformerId>() shall invoke E2E_PXXProtect(), passing to that function the appropriate Config and State structures (see [SWS_E2EXf_00125] and [SWS_E2EXf_00126]) that are associated with <transformerId>, as well as buffer and bufferLength (only for P04, P05, P06, P07, P11 and P22) that were updated in above requirements SWS_E2EXf_00108, SWS_E2EXf_00109, SWS_E2EXf_00115, SWS_E2EXf_00111.] (RS_E2E_08538)

[SWS_E2EXf_00167] {DRAFT} [If DataTransformationStatusForwarding is set to transformerStatusForwarding, the function E2EXf_<transformerId>() shall invoke E2E_PXXForward(), passing to that function the appropriate Config and State structures (see [SWS_E2EXf_00125] and [SWS_E2EXf_00126]) that are associated with <transformerId>, as well as buffer and bufferLength (only for P04, P05, P06, P07, P11 and P22) that were updated in above requirements SWS_E2EXf_00108, SWS_E2EXf_00109, SWS_E2EXf_00115, SWS_E2EXf_00111. In addition the parameter ForwardStatus shall be passed to the E2E_PXXForward() function, which is provided by the RTE. This parameter is associated with the optional IN parameter forwardStatus from Rte_Write (SWS_Rte_01071), Rte_Send (SWS_Rte_01072), Rte_IWrite (SWS_Rte_03744) and Rte_IWriteRef (SWS_Rte_05509)] (RS_E2E_08538)

The parameter DataTransformationStatusForwarding is set as defined in the Software Component Template [11].

[SWS_E2EXf_00018] In case E2E_PXXProtect() or E2E_PXXForward() returns E2E_E_OK, then E2EXf_<transformerId> shall return E_OK, otherwise E2EXf_<transformerId> shall return E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

7.6.3 E2EXf_Inv_<transformerId>(check-function)

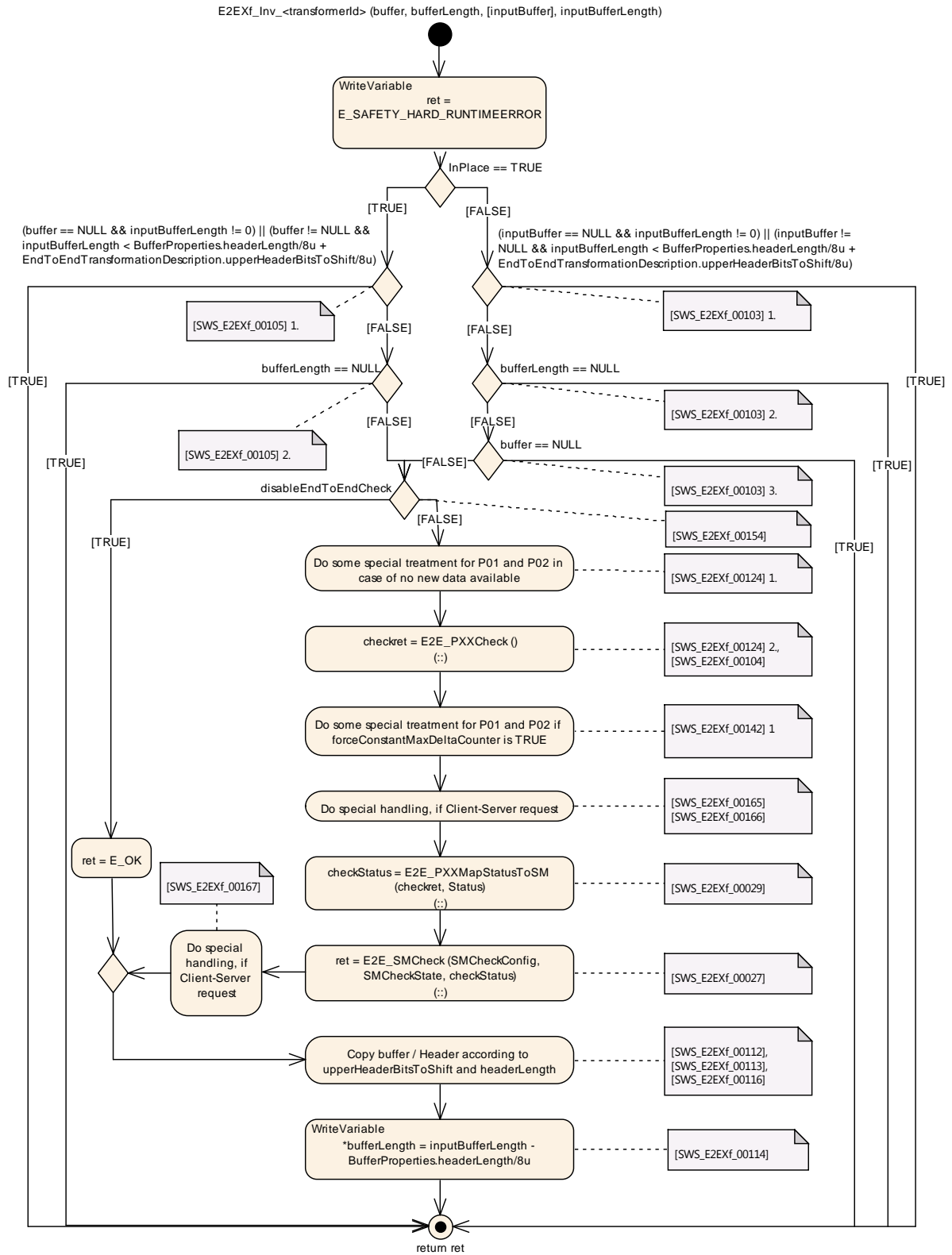


Figure 7-4: E2EXf_Inv_<transformerId>function overview

Figure above provides an activity diagram of the functionality provided by the API function E2EXf_Inv_<transformerId>.

[SWS_E2EXf_00025] The function E2EXf_Inv_<transformerId> shall be generated for each E2E-protected data element (<transformerId>).] (RS_E2E_08538)

[SWS_E2EXf_00105] In-place E2EXf_Inv_<transformerId> shall perform the following two precondition checks, without continuing further processing:

1. (buffer == NULL && inputBufferLength != 0)
||
(buffer != NULL && inputBufferLength < BufferProperties.headerLength/8u + EndToEndTransformationDescription.upperHeaderBitsToShift/8u)
2. bufferLength == NULL.

If any of above conditions is TRUE, then the function shall return E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

[SWS_E2EXf_00103] Out-of-place E2EXf_Inv_<transformerId> shall perform the following three precondition checks, without continuing further processing:

1. (inputBuffer == NULL && inputBufferLength != 0)
||
(inputBuffer != NULL && inputBufferLength < BufferProperties.headerLength/8u + EndToEndTransformationDescription.upperHeaderBitsToShift/8u)
2. If (bufferLength == NULL)
3. If (buffer == NULL).

If any of above conditions is TRUE, then the function shall return E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

Depending on the configuration of the E2EXf regarding buffer operations (in-place and out-of-place), a valid pointer to the output buffer, but the length is too small may result in a soft or hard error, depending on E2EXf configuration and actual given length:

Condition	Return Value	Description
Valid pointer to the output buffer, but too small length to perform buffer operation.	E_SAFETY_HARD_RUNTIMEERROR	cannot perform header copy operation, leaving buffer with invalid data according to configuration. Hard runtime error must be returned.
Valid pointer to the output buffer and sufficient length for buffer operation, E2E profile 1 or 2, and buffer length not matching configured data length.	E_SAFETY_HARD_RUNTIMEERROR	E2E profile 1 and 2 are not performing data length checks on their own, and are expected to be used with ComXf with fixed length messages (so this

		scenario is not expected to happen during normal operation).
Valid pointer to the output buffer and sufficient length for buffer operation on data, but length too small for E2E profile configuration (all E2E profiles except 1 or 2)	E_SAFETY_VALID_ERR, E_SAFETY_NODATA_ERR, E_SAFETY_INIT_ERR, E_SAFETY_INVALID_ERR	Depending on the internal state of the E2E state machine, one of the checks returns an E2E-error.

[SWS_E2EXf_00167] In case of Client/Server-Communication on the server side, if the ret equals to E_SAFETY_*_ERR, the value shall be overwritten to E_SAFETY_HARD_RUNTIMEERROR.](RS_E2E_08538)

The handling of E_SAFETY_*_ERR errors as hard errors creates a TransformerHardErrorEvent. The intended use of this event, is to inform the SWC, that there was a failed Server-Client Call and possible some Error Handling has to be done.

Note that the function E2EXf_Inv_<transformerId> may be realized by a plain function, inline function or a macro (implementation-specific). The functions E2EXf_Inv_<transformerId> may call some internal common functions.

[SWS_E2EXf_00140] For PXX = 01 or 02 (i.e. for profile 1 and 2), the out-of-place function E2EXf_Inv_<transformerId> shall

1. if(inputBuffer == NULL and inputBufferLength == 0), then
 - variable NewDataAvailable of state object of type E2E_PXXCheckStateType (see [SWS_E2EXf_00125]) associated with <transformerId> shall be set to FALSE
2. else if (inputBufferLength == config->DataLength/8), then
 - variable NewDataAvailable of state object of type E2E_PXXCheckState Type (see [SWS_E2EXf_00125]) associated with <transformerId> shall be set to TRUE.

else return E_SAFETY_HARD_RUNTIMEERROR.](RS_E2E_08538)

[SWS_E2EXf_00123] For PXX = 01 or 02 (i.e. for profiles 1 and 2), the out-of-place function E2EXf_Inv_<transformerId> shall invoke E2E_PXXCheck(), passing to that function:

- Config,
- State,
- Data

Concerning pointer to Data: if(inputBuffer == NULL and inputBufferLength == 0), then it shall pass a pointer to a 1-byte variable of E2E transformer, otherwise it shall pass inputBuffer.](RS_E2E_08538)

[SWS_E2EXf_00141] For PXX = 01 or 02 (i.e. for profiles 1 and 2), the in-place function E2EXf_Inv_<transformerId> shall

1. If (buffer == NULL and inputBufferLength == 0), then
 - variable NewDataAvailable of state object of type E2E_PXXCheckStateType (see [SWS_E2EXf_00125]) associated with <transformerId> shall be set to FALSE.
2. Else if (inputBufferLength == config->DataLength/8), then
 - variable NewDataAvailable of state object of type E2E_PXXCheckState Type (see [SWS_E2EXf_00125]) associated with <transformerId> shall be set to TRUE.
3. Else return E_SAFETY_HARD_RUNTIMEERROR.

] (RS_E2E_08538)

[SWS_E2EXf_00124] For PXX = 01 or 02 (i.e. for profiles 1 and 2), the in-place function E2EXf_Inv_<transformerId> shall invoke E2E_PXXCheck(), passing to that function:

- Config,
- State,
- Data

Concerning pointer to Data: if (buffer == NULL and inputBufferLength == 0), then it shall pass a pointer to a 1-byte variable of E2E transformer, otherwise it shall pass buffer.] (RS_E2E_08538)

[SWS_E2EXf_00142] If configuration parameter profileBehavior is PRE_R4_2, then for PXX = 01 or 02, E2EXf_Inv_<transformerId> () shall set the variable MaxDeltaCounter of the state object to the value of variable MaxDeltaCounterInit of the corresponding configuration object.] (RS_E2E_08538)

SWS_E2EXf_00123 and SWS_E2EXf_00124 either pass the pointer to valid buffer containing the E2E-protected data (in case data is available / is received) or otherwise provide a pointer to a dummy local variable, which is anyway not used by the E2E checks (NewDataAvailable is set to FALSE at the same time). Additionally, the length of the Buffer is checked, which is not done by profiles 1 and 2. It is necessary because the profiles P1 and P2 behave different from the newer profiles 4, 5 and 6. Profile 1 and 2 do not accept a NULL pointer, and they provide a sophisticated dynamic MaxDeltaCounter and re-synchronization mechanism different to the less complex checks provided in the newer profiles.

However, changes in the legacy profiles are not done to keep full backward-compatibility with existing implementations. Therefore, the new configuration parameter profileBehavior configures the E2EXf to reset the MaxDeltaCounter after each call of E2E_PXXCheck(), and the provided recommended configuration values for MaxNoNewOrRepeatedData and SyncCounterInit together with the different behavior of the mapping function E2E_PXXMapStatusToSM enforce a common behavior of all profiles when combined with the E2E state machine.

[SWS_E2EXf_00104] For PXX = 04, 05, 06, 07, 11, 22: the function E2EXf_Inv_<transformerId> shall invoke E2E_PXXCheck(), passing to that function:

- config,
- state,
- data length: inputBufferLength

pointer to data:inputBuffer (out-of-place version) or buffer (in-place version).] (RS_E2E_08538)

[SWS_E2EXf_00029] The function E2EXf_Inv_<transformerId> shall invoke E2E_PXXMapStatusToSM(), passing to that function the return value of E2E_PXXCheck and the profile's check Status (variable Status of state object of type E2E_PXXCheckStateType, see [SWS_E2EXf_00125]), to obtain the profile-independent check status. For P1/P2 mapping functions, there is an additional call parameter profileBehavior:

- if configuration parameter profileBehavior is R4_2, then E2E_PXXMapStatusToSM() shall be invoked with the call parameter profileBehavior = 1
- if configuration parameter profileBehavior is PRE_R4_2, then E2E_PXXMapStatusToSM() shall be invoked with call parameter profileBehavior = 0] (RS_E2E_08538)

[SWS_E2EXf_00028] The function E2EXf_Inv_<transformerId> shall invoke the E2E_SMCheck() function, passing to that function the configuration object of type E2E_SMConfigType (see [SWS_E2EXf_00126] and [SWS_E2EXf_00088]) and state object of type E2E_SMCheckStateType (see [SWS_E2EXf_00125]) that are associated with <transformerId>, plus the profile-independent check status that was computed by E2E_PXXMapStatusToSM() in SWS_E2EXf_00029.] (RS_E2E_08538)

[SWS_E2EXf_00027] If E2E_SMCheck() returns E2E_E_OK, then:

- the high nibble of the return of the function E2EXf_Inv_<transformerId> shall be set to the low nibble of the state of the state machine (member SMState of object of type E2E_SMStateType that is associated with <transformerId>, see [SWS_E2EXf_00125]).
- The low nibble of the return of the function E2EXf_Inv_<transformerId> shall be set to the low nibble of the profile-independent check status of type E2E_PCheckStatusType.

If E2E_SMCheck() does not return E2E_E_OK, the return value shall be E_SAFETY_SOFT_RUNTIMEERROR.] (RS_E2E_08538)

[SWS_E2EXf_00112] If (buffer != NULL &&EndToEndTransformationDescription.upperHeaderBitsToShift> 0), in-place E2EXf_Inv_<transformerId> shall copy the first upper HeaderBitsToShiftbits, in parameter buffer, in direction right by "distance" of BufferProperties.headerLength.] (RS_E2E_08538)

[SWS_E2EXf_00113] If (inputBuffer != NULL &&EndToEndTransformationDescription.upperHeaderBitsToShift> 0), out-of-place E2EXf_Inv_<transformerId> shall copy the first upper HeaderBitsToShiftbits from inputBuffer to buffer, and then copy the remaining part of inputBuffer skipping E2E header (i.e. starting with offset upperHeaderBitsToShift+BufferProperties.headerLength) to parameter buffer starting with the destination offset of (upperHeaderBitsToShift).] (RS_E2E_08538)

[SWS_E2EXf_00116] If (inputBuffer != NULL &&EndToEndTransformationDescription.upperHeaderBitsToShift == 0), out-of-place E2EXf_Inv_<transformerId> shall copy inputBuffer starting with the offset of BufferProperties.headerLength, to buffer.] (RS_E2E_08538)

[SWS_E2EXf_00114] If inputBufferLength == 0, then E2EXf_Inv_<transformerId> shall set *bufferLength = 0, otherwise it shall set *bufferLength = inputBufferLength - BufferProperties.headerLength/8.] (RS_E2E_08538)

The case where inputBufferLength is > 0 but shorter than header is covered by [SWS_E2EXf_00105].

7.6.4 De-Initialization

[SWS_E2EXf_00148] E2EXf_Delnit() shall check shall set the module initialization state to FALSE.] (RS_E2E_08538)

7.7 Error classification

7.7.1 Development Errors

[SWS_E2EXf_00137] The E2E Transformer shall be able to detect the following development errors:

Type of error	Related error code	Value [hex]
Error code if any other API service, except GetVersionInfo is called before the transformer module was initialized with Init or after a call to Delnit	E2EXF_E_UNINIT	0x01
Error code if an invalid configuration set was selected	E2EXF_E_INIT_FAILED	0x02
API service called with wrong parameter	E2EXF_E_PARAM	0x03
API service called with invalid pointer	E2EXF_E_PARAM_POINTER	0x04

] (RS_E2E_08538)

[SWS_E2EXf_00144] If the XfrmDevErrorDetect switch is enabled and the configuration variant is VARIANT-POST-BUILD, the function E2EXf_Init shall check if a NULL pointer is passed for the ConfigPtr parameter. In case of an error the remaining function shall not be executed and the function E2EXf_Init shall report development error code E2EXF_E_PARAM_POINTER to the Det_ReportError service of the Default Error Tracer.] (RS_E2E_08538)

[SWS_E2EXf_00145] If the XfrmDevErrorDetect switch is enabled and the configuration variant is VARIANT-POST-BUILD, the function E2EXf_Init shall check the contents of the given configuration set for being within the allowed boundaries. If the function E2EXf_Init detects an error, then it shall skip the initialization of the E2E Transformer, keep the module internal state as uninitialized and it shall report development error code E2EXF_E_INIT_FAILED to the Det_ReportError service of the Default Error Tracer.] (RS_E2E_08538)

[SWS_E2EXf_00146] If the configuration parameter XfrmDevErrorDetect is enabled, the function E2EXf_DeInit shall check if the E2E transformer is initialized. In case of an error, the function E2EXf_DeInit shall return without any effect and shall report the error to the Default Error Tracer with the error code E2EXF_E_UNINIT.] (RS_E2E_08538)

[SWS_E2EXf_00149] If the XfrmDevErrorDetect switch is enabled, the function E2EXf_GetVersionInfo shall check if a NULL pointer is passed for the VersionInfo parameter. In case of an error the remaining function E2EXf_GetVersionInfo shall not be executed and the function E2EXf_GetVersionInfo shall report development error code E2EXF_E_PARAM_POINTER to the Det_ReportError service of the Default Error Tracer.
] (RS_E2E_08538)

[SWS_E2EXf_00150] If the configuration parameter XfrmDevErrorDetect is enabled, all parameters of API E2EXf_<transformerId> (see SWS_E2EXf_00032) shall be checked for being in the allowed range. In case of an error the mode switch shall not be executed, the error shall be reported to the Default Error Tracer with the value E2EXF_E_PARAM resp. E2EXF_E_PARAM_POINTER in case of a pointer argument and the routine shall return the value E_SAFETY_HARD_RUNTIMEERROR.
] (RS_E2E_08538)

[SWS_E2EXf_00151] If the configuration parameter XfrmDevErrorDetect is enabled, the API E2EXf_<transformerId> (see SWS_E2EXf_00032) shall check if the E2E Transformer is initialized. In case of an error the routine shall not be executed, the error shall be reported to the Default Error Tracer with the error code E2EXF_E_UNINIT and the routine shall return the value E_SAFETY_HARD_RUNTIMEERROR.
] (RS_E2E_08538)

[SWS_E2EXf_00152] If the configuration parameter XfrmDevErrorDetect is enabled, all parameters of API E2EXf_Inv_<transformerId> (see SWS_E2EXf_00034) shall be checked for being in the allowed range. In case of an error the mode switch shall not be executed, the error shall be reported to the Default Error Tracer with the value E2EXF_E_PARAM resp. E2EXF_E_PARAM_POINTER in case of a pointer argument and the routine shall return the value E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

[SWS_E2EXf_00153] If the configuration parameter XfrmDevErrorDetect is enabled, the API E2EXf_Inv_<transformerId> (see SWS_E2EXf_00034) shall check

if the E2E Transformer is initialized. In case of an error the routine shall not be executed, the error shall be reported to the Default Error Tracer with the error code E2EXF_E_UNINIT and the routine shall return the value E_SAFETY_HARD_RUNTIMEERROR.] (RS_E2E_08538)

7.7.2 Runtime Errors

[SWS_E2EXf_00122] The runtime errors detected by the E2EXf_<transformerId> function shall be reported as return value to the caller (i.e. to RTE).] (RS_E2E_08538)

[SWS_E2EXf_00009] The runtime errors detected by E2EXf_Inv_<transformerId> function and errors in the protected E2E communication shall be reported as return value to the caller (i.e. to RTE).] (RS_E2E_08538)

7.7.3 Transient Faults

There are no Transient Faults reported by E2E Transformer.

7.7.4 Production Errors

There are no production errors reported to DEM by E2E Transformer, because the error information is returned synchronously to the caller (RTE), which is then forwarded to calling software component.

7.7.5 Extended Production Errors

All Extended Production Errors valid for E2E Transformer are specified in ASWS Transformer General [9].

8 API specification

8.1 Imported types

In this chapter, all types included from the following modules are listed:

[SWS_E2EXf_00047]

<i>Module</i>	<i>Header File</i>	<i>Imported Type</i>
E2E	E2E.h	E2E_P01CheckStateType
	E2E.h	E2E_P01CheckStatusType
	E2E.h	E2E_P01ConfigType
	E2E.h	E2E_P01DataIDMode
	E2E.h	E2E_P01ProtectStateType
	E2E.h	E2E_P02CheckStateType
	E2E.h	E2E_P02CheckStatusType
	E2E.h	E2E_P02ConfigType
	E2E.h	E2E_P02ProtectStateType
	E2E.h	E2E_P04CheckStateType
	E2E.h	E2E_P04CheckStatusType
	E2E.h	E2E_P04ConfigType
	E2E.h	E2E_P04ProtectStateType
	E2E.h	E2E_P05CheckStateType
	E2E.h	E2E_P05CheckStatusType
	E2E.h	E2E_P05ConfigType
	E2E.h	E2E_P05ProtectStateType
	E2E.h	E2E_P06CheckStateType
	E2E.h	E2E_P06CheckStatusType
	E2E.h	E2E_P06ConfigType
	E2E.h	E2E_P06ProtectStateType
	E2E.h	E2E_P07CheckStateType
	E2E.h	E2E_P07CheckStatusType
	E2E.h	E2E_P07ConfigType

	E2E.h	E2E_P07ProtectStateType
	E2E.h	E2E_P11CheckStateType
	E2E.h	E2E_P11CheckStatusType
	E2E.h	E2E_P11ConfigType
	E2E.h	E2E_P11DataIDMode
	E2E.h	E2E_P11ProtectStateType
	E2E.h	E2E_P22CheckStateType
	E2E.h	E2E_P22CheckStatusType
	E2E.h	E2E_P22ConfigType
	E2E.h	E2E_P22ProtectStateType
	E2E.h	E2E_PCheckStatusType
	E2E.h	E2E_SMCheckStateType
	E2E.h	E2E_SMConfigType
	E2E.h	E2E_SMStateType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

]()

Furthermore, ASWS Transformer General [9] defines types which shall be imported.

8.2 Type definitions

8.2.1 E2EXf_ConfigType

[SWS_E2EXf_00030]

Name	E2EXf_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	--
	Comment	--
Description	Parent container for the configuration of E2E Transformer. The content is implementation-specific.	
Available via	E2EXf.h	

J(RS_E2E_08538)

8.3 Function definitions

8.3.1 E2EXf_<transformerId>

[SWS_E2EXf_00032]

Service Name	E2EXf_<transformerId>	
Syntax	<pre>uint8 E2EXf_<transformerId> (uint8* buffer, uint32* bufferLength, [const uint8* inputBuffer], uint32 inputBufferLength)</pre>	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	input Buffer	This argument only exists for E2E transformers configured for out-of-place transformation. It holds the input data for the transformer.
	input Buffer Length	This argument holds the length of the E2E transformer's input data (in the inputBuffer argument).
Parameters (inout)	buffer	<p>This argument is only an INOUT argument for E2E transformers which are configured for in-place transformation. This is the buffer where the E2E transformer places its output data. If the E2E transformer is configured for in-place transformation, it also contains its input data. If the E2E transformer uses in-place transformation and has a header Length different from 0, the output data of the previous transformer begin at position headerLength.</p> <p>This argument is only an OUT argument for E2E transformers configured for out-of-place transformation. It is the buffer allocated by the RTE, where the transformed data has to be stored by the transformer.</p>
Parameters (out)	buffer Length	Used length of the buffer
Return value	uint8	<p>0x00 (E_OK): Function performed successfully.</p> <p>0xFF (E_SAFETY_HARD_RUNTIMEERROR): A runtime error occurred, safety properties could not be checked and no output data could be produced.</p>
Description	Protects the array/buffer to be transmitted, using the in-place transformation.	

Available via	E2EXf.h
----------------------	---------

J(RS_E2E_08538)

The return codes of E2EXf_<transformerId> are specified in TransformerTypes, see AWS Transformer General.

8.3.2 E2EXf_Inv_<transformerId>

[SWS_E2EXf_00034]

Service Name	E2EXf_Inv_<transformerId>	
Syntax	<pre>uint8 E2EXf_Inv_<transformerId> (uint8* buffer, uint32* bufferLength, [const uint8* inputBuffer], uint32 inputBufferLength)</pre>	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	input Buffer	This argument only exists for E2E transformers configured for out-of-place transformation. It holds the input data for the transformer. If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, it will hand over a NULL pointer to the transformer.
	input Buffer Length	This argument holds the length of the transformer's input data (in the input Buffer argument). If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, the length will be equal to 0.
Parameters (inout)	buffer	This argument is only an INOUT argument for E2E transformers, which are configured for in-place transformation. It is the buffer where the input data are placed by the RTE and which is filled by the transformer with its output. If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, it will hand over a NULL pointer to the transformer. This argument is only an OUT argument for E2E transformers configured for out-of-place transformation. It is the buffer allocated by the RTE, where the transformed data has to be stored by the transformer.
Parameters (out)	buffer Length	Used length of the output buffer.
Return value	uint8	The high nibble represents the state of the E2E state machine, the low nibble represents the status of the last E2E check. For the following return codes, please see SWS Transformer General:

	<p>0x00 (E_OK) This means VALID_OK 0x01 (E_SAFETY_VALID_REP) 0x02 (E_SAFETY_VALID_SEQ) 0x03 (E_SAFETY_VALID_ERR) 0x05 (E_SAFETY_VALID_NND) 0x20 (E_SAFETY_NODATA_OK) 0x21 (E_SAFETY_NODATA_REP) 0x22 (E_SAFETY_NODATA_SEQ) 0x23 (E_SAFETY_NODATA_ERR) 0x25 (E_SAFETY_NODATA_NND) 0x30 (E_SAFETY_INIT_OK) 0x31 (E_SAFETY_INIT_REP) 0x32 (E_SAFETY_INIT_SEQ) 0x33 (E_SAFETY_INIT_ERR) 0x35 (E_SAFETY_INIT_NND) 0x40 (E_SAFETY_INVALID_OK) 0x41 (E_SAFETY_INVALID_REP) 0x42 (E_SAFETY_INVALID_SEQ) 0x43 (E_SAFETY_INVALID_ERR) 0x45 (E_SAFETY_INVALID_NND) 0x77 (E_SAFETY_SOFT_RUNTIMEERROR) A runtime error occurred, safety properties could not be checked (state or status cannot be determined) but non-protected output data could be produced nonetheless. 0xFF (E_SAFETY_HARD_RUNTIMEERROR): A runtime error occurred, safety properties could not be checked and no output data could be produced.</p>
Description	Checks the received data. If the data can be used by the caller, then the function returns E_OK.
Available via	E2EXf.h

](RS_E2E_08538)

The return codes of E2EXf_Inv_<transformerId> are specified in TransformerTypes, see ASWS Transformer General.

8.3.3 E2EXf_Init

Add the following function:

[SWS_E2EXf_00035]

Service Name	E2EXf_Init
Syntax	<pre>void E2EXf_Init (const E2EXf_ConfigType* config)</pre>
Service ID [hex]	0x01
Sync/Async	Synchronous

Reentrancy	Reentrant	
Parameters (in)	config	Pointer to a selected configuration structure, in the post-build-selectable variant. NULL in link-time variant.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the state of the E2E Transformer. The main part of it is the initialization of the E2E library state structures, which is done by calling all init-functions from E2E library.	
Available via	E2EXf.h	

](RS_E2E_08538)

8.3.4 E2EXf_DeInit

[SWS_E2EXf_00138]

Service Name	E2EXf_DeInit
Syntax	<pre>void E2EXf_DeInit (void)</pre>
Service ID [hex]	0x02
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	Deinitializes the E2E transformer.
Available via	E2EXf.h

](RS_E2E_08538)

8.3.5 E2EXf_GetVersionInfo

[SWS_E2EXf_00036]

Service Name	E2EXf_GetVersionInfo	
Syntax	<pre>void E2EXf_GetVersionInfo (Std_VersionInfoType* versioninfo)</pre>	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module.
Return value	None	
Description	Returns the version information of this module.	
Available via	E2EXf.h	

](RS_E2E_08538)

8.4 Call-back notifications

None

8.5 Scheduled functions

None

8.6 Expected Interfaces

In this chapter all external interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all external interfaces, which are required to fulfill the core functionality of the module.

[SWS_E2EXf_00037]

API Function	Header File	Description
E2E_P01-Check	E2E.h	Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.

E2E_P01-CheckInit	E2E.h	Initializes the check state
E2E_P01-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 01. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P01-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 1 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 1 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P01-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID.
E2E_P01-ProtectInit	E2E.h	Initializes the protection state.
E2E_P02-Check	E2E.h	Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.
E2E_P02-CheckInit	E2E.h	Initializes the check state
E2E_P02-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 02. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P02-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 2 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 2 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P02-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.
E2E_P02-ProtectInit	E2E.h	Initializes the protection state.
E2E_P04-Check	E2E.h	Checks the Data received using the E2E profile 4. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.
E2E_P04-CheckInit	E2E.h	Initializes the check state
E2E_P04-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 04. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P04-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 4 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 4 delivers a more fine-granular status, but this is not relevant for the E2E state machine.

E2E_P04-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 4. This includes checksum calculation, handling of counter and Data ID.
E2E_P04-ProtectInit	E2E.h	Initializes the protection state.
E2E_P05-Check	E2E.h	Checks the Data received using the E2E profile 5. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.
E2E_P05-CheckInit	E2E.h	Initializes the check state
E2E_P05-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 05. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P05-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 5 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 5 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P05-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 5. This includes checksum calculation, handling of counter.
E2E_P05-ProtectInit	E2E.h	Initializes the protection state.
E2E_P06-Check	E2E.h	Checks the Data received using the E2E profile 6. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.
E2E_P06-CheckInit	E2E.h	Initializes the check state
E2E_P06-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 06. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P06-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 6 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 6 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P06-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 6. This includes checksum calculation, handling of counter.
E2E_P06-ProtectInit	E2E.h	Initializes the protection state.
E2E_P07-Check	E2E.h	Checks the Data received using the E2E profile 7. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.
E2E_P07-	E2E.h	Initializes the check state

CheckInit		
E2E_P07-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 07. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P07-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 7 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 7 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P07-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 7. This includes checksum calculation, handling of counter and Data ID.
E2E_P07-ProtectInit	E2E.h	Initializes the protection state.
E2E_P11-Check	E2E.h	Checks the Data received using the E2E profile 11. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.
E2E_P11-CheckInit	E2E.h	Initializes the check state
E2E_P11-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 11. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P11-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 11 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 11 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P11-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 11. This includes checksum calculation, handling of counter.
E2E_P11-ProtectInit	E2E.h	Initializes the protection state.
E2E_P22-Check	E2E.h	Checks the Data received using the E2E profile 22. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.
E2E_P22-CheckInit	E2E.h	Initializes the check state
E2E_P22-Forward	E2E.h	Protects data which is forwarded by using the E2E profile 22. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. Tags: atp.Status=draft
E2E_P22-MapStatus-ToSM	E2E.h	The function maps the check status of Profile 22 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 22 delivers a more fine-granular status, but this is not relevant for the E2E state machine.

E2E_P22-Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 22. This includes checksum calculation, handling of counter.
E2E_P22-ProtectInit	E2E.h	Initializes the protection state.
E2E_SM-Check	E2E.h	Checks the communication channel. It determines if the data can be used for safety-related application, based on history of checks performed by a corresponding E2E_P0XCheck() function.
E2E_SM-CheckInit	E2E.h	Initializes the state machine.

](RS_E2E_08538)

8.6.2 Optional Interfaces

None

8.6.3 Configurable interfaces

None

9 Sequence diagrams

9.1 Protect – E2EXf_<transformerId>

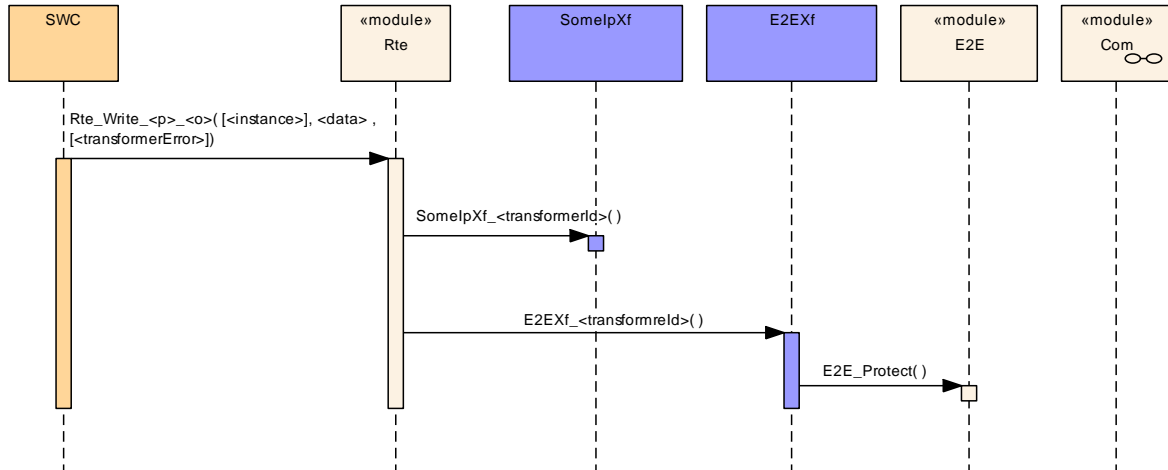


Figure 9-1: E2EXf

9.2 Check – E2EXf_Inv_<transformerId>

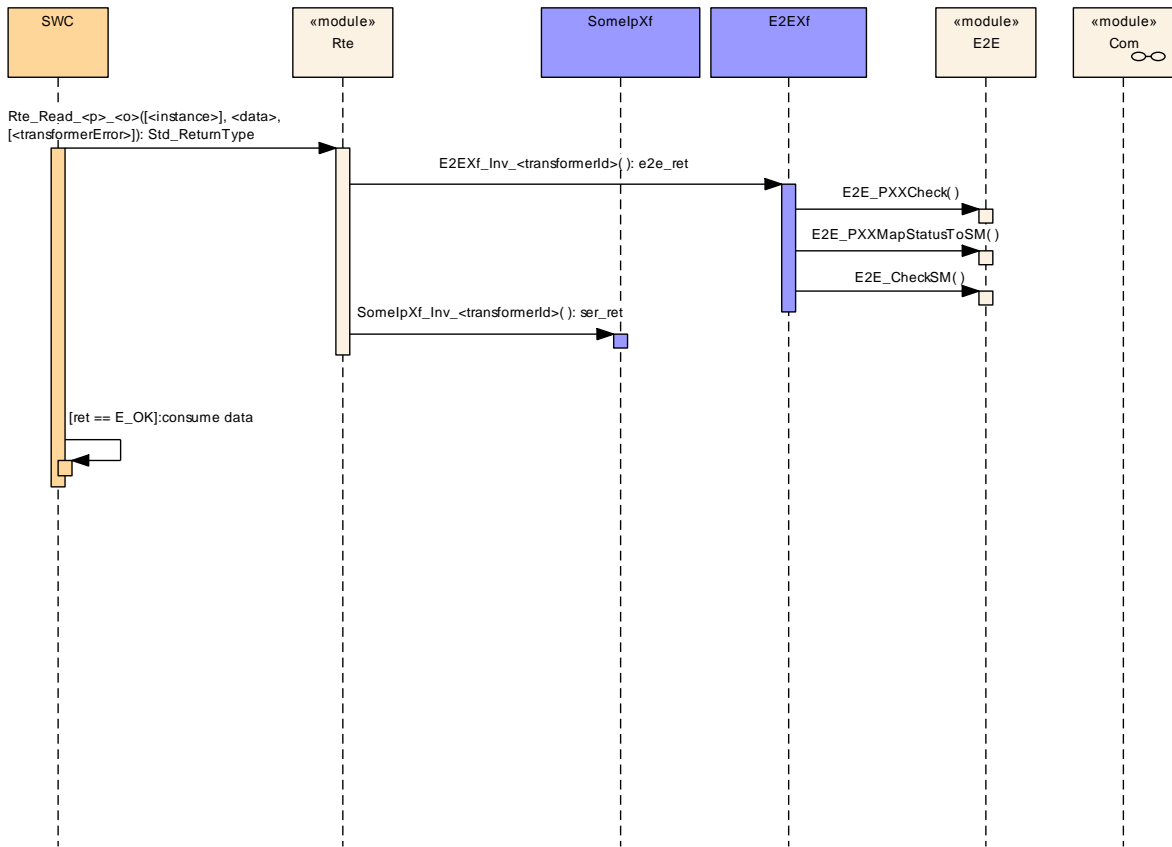


Figure 9-2: E2EXf_Inv

10 Configuration specification

There is no module specific ECU configuration for E2E Transformer. The following is used for the generation of E2E transformer:

1. Options defined in TPS System Template (defining functional options related to protection, e.g. IDs, counters)
2. Options defined in TPS Software Component template (defining options for specific ports that override options defined in TPS System Template)
3. Options defined in ASWS Transformer General (Mapping of TransformationTechnology entities of a DataTransformation to implementing BswModuleEntry entities).

In order to avoid redundancy and inconsistency, the configuration is not repeated here.

[SWS_E2EXf_00156] The apiServicePrefix of the E2E Transformer's EcuC shall be set to E2EXf.] (SRS_BSW_00159)

11 Not applicable requirements

-