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	R22-11

Document Change History

Date	Release	Changed by	Description
2022-11-24	R22-11	AUTOSAR Release Management	• Remove duplicated requirement SWS_E2EXf_00195 • Correction of transformer call in SWS_E2EXf_00203, SWS_E2EXf_00196 • Use consistent function names (e.g. E2EXf_handling_PXXm_server, E2EXf_handling_PXXm_client) • Counter handling for client/server communication (methods) updated • Correction of transformer status forwarding • Correction of MaxDataLength and Offset in profile P07m
2021-11-25	R21-11	AUTOSAR Release Management	• Added Concept 700 text and figures (E2E for fields) • Added Description of Profile 8m and 44m (E2E for methods)
2020-11-30	R20-11	AUTOSAR Release Management	• Added Description of Profile 4m and 7m (E2E for methods) • Updated added drawings of functions • Updated API Specification

2019-11-28	R19-11	AUTOSAR Release Management	• 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	• Fix routine prototypes to correctly list optional parameters. • Correction applicable configuration parameter for data length 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	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Added support for Profiles P7, P11, P22 • Various minor improvements
2015-07-31	4.2.2	AUTOSAR Release Management	• Various minor fixes
2014-10-31	4.2.1	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

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:

1. On sender side, one data element serialized to one I-signal, and protected with one E2E-protection
2. 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.
3. On receiver side, one I-signal checked:
    - Once: resulting with one data element (i.e. no fan-out)
    - Several times, with the same settings, but by independent functions (e.g. by ASIL-independent receiver SW-Cs),
    - 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,
    - 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 & related standards and norms

[1] General Specification of Transformers
AUTOSAR_ASWS_TransformerGeneral

[2] E2E Protocol Specification
AUTOSAR_PRS_E2EProtocol

[3] Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_SWS_E2ELibrary

[4] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[5] System Template
AUTOSAR_TPS_SystemTemplate

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[7] SOME/IP Protocol Specification
AUTOSAR_PRS_SOMEIPProtocol

3.2 Related specification

AUTOSAR provides a General Specification on Transformers [1, ASWS Transformer General], which is also valid for E2E Transformer.

Thus, the specification [1, ASWS Transformer General] shall be considered as additional and required specification for E2E Transformer.

4 Constraints and assumptions

4.1 Limitations

General limitations regarding E2E protection and the detectable failure modes are described in [2, PRS E2EProtocol].

Further, the following limitations are known:

1. Error reporting to DEM not yet specified.

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 [3, SWS E2ELibrary]. The E2E Library provides data types and stateless functions. E2E Transformer executes the E2E Library routines passing the configuration and state as function parameters.

5.1 Supported configuration variants

There are currently no explicit Pre-compile time configuration settings apart from the settings specified in [4, SWS BSW General] and [1, ASWS Transformer].

6 Requirements Tracing

Requirement	Description	Satisfied by
[RS_E2E_08538]	An E2E Transformer shall be provided	[SWS_E2EXf_00009] [SWS_E2EXf_00011] [SWS_E2EXf_00018] [SWS_E2EXf_00020] [SWS_E2EXf_00021] [SWS_E2EXf_00023] [SWS_E2EXf_00024] [SWS_E2EXf_00025] [SWS_E2EXf_00026] [SWS_E2EXf_00027] [SWS_E2EXf_00028] [SWS_E2EXf_00029] [SWS_E2EXf_00030] [SWS_E2EXf_00032] [SWS_E2EXf_00034] [SWS_E2EXf_00035] [SWS_E2EXf_00036] [SWS_E2EXf_00037] [SWS_E2EXf_00048] [SWS_E2EXf_00087] [SWS_E2EXf_00088] [SWS_E2EXf_00089] [SWS_E2EXf_00090] [SWS_E2EXf_00096] [SWS_E2EXf_00102] [SWS_E2EXf_00103] [SWS_E2EXf_00104] [SWS_E2EXf_00105] [SWS_E2EXf_00106] [SWS_E2EXf_00107] [SWS_E2EXf_00108] [SWS_E2EXf_00109] [SWS_E2EXf_00111] [SWS_E2EXf_00112] [SWS_E2EXf_00113] [SWS_E2EXf_00114] [SWS_E2EXf_00115] [SWS_E2EXf_00116] [SWS_E2EXf_00118] [SWS_E2EXf_00119] [SWS_E2EXf_00120] [SWS_E2EXf_00122] [SWS_E2EXf_00123] [SWS_E2EXf_00124] [SWS_E2EXf_00125] [SWS_E2EXf_00126] [SWS_E2EXf_00130] [SWS_E2EXf_00132] [SWS_E2EXf_00133] [SWS_E2EXf_00134] [SWS_E2EXf_00137] [SWS_E2EXf_00138] [SWS_E2EXf_00139] [SWS_E2EXf_00140] [SWS_E2EXf_00141] [SWS_E2EXf_00142] [SWS_E2EXf_00144] [SWS_E2EXf_00145] [SWS_E2EXf_00146] [SWS_E2EXf_00148] [SWS_E2EXf_00149] [SWS_E2EXf_00150] [SWS_E2EXf_00151] [SWS_E2EXf_00152] [SWS_E2EXf_00153] [SWS_E2EXf_00154] [SWS_E2EXf_00155] [SWS_E2EXf_00158] [SWS_E2EXf_00159] [SWS_E2EXf_00161] [SWS_E2EXf_00162] [SWS_E2EXf_00163] [SWS_E2EXf_00164] [SWS_E2EXf_00165] [SWS_E2EXf_00166] [SWS_E2EXf_00167] [SWS_E2EXf_00168] [SWS_E2EXf_00169] [SWS_E2EXf_00170] [SWS_E2EXf_00171] [SWS_E2EXf_00172] [SWS_E2EXf_00173] [SWS_E2EXf_00175] [SWS_E2EXf_00176] [SWS_E2EXf_00177] [SWS_E2EXf_00178] [SWS_E2EXf_00179] [SWS_E2EXf_00180] [SWS_E2EXf_00181] [SWS_E2EXf_00182] [SWS_E2EXf_00183] [SWS_E2EXf_00184] [SWS_E2EXf_00185] [SWS_E2EXf_00186] [SWS_E2EXf_00187] [SWS_E2EXf_00188] [SWS_E2EXf_00189] [SWS_E2EXf_00190] [SWS_E2EXf_00191] [SWS_E2EXf_00192] [SWS_E2EXf_00193] [SWS_E2EXf_00194] [SWS_E2EXf_00196] [SWS_E2EXf_00197] [SWS_E2EXf_00198] [SWS_E2EXf_00199] [SWS_E2EXf_00200] [SWS_E2EXf_00201] [SWS_E2EXf_00202] [SWS_E2EXf_00203] [SWS_E2EXf_00204] [SWS_E2EXf_00205] [SWS_E2EXf_00206] [SWS_E2EXf_00207] [SWS_E2EXf_00208]

△

Requirement	Description	Satisfied by
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_E2EXf_00156]
[SWS_Xfrm_00071]		[SWS_E2EXf_NA_00001]

Table 6.1: RequirementsTracing

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, forward, state machine). E2E transformer invokes E2E 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 specific ECU configuration because its whole configuration is based on the E2ETransformationDescription, the E2ETransformationISignal Props and the E2ETransformationComSpecProps. Thus the generic ECU configuration of the [1, ASWS Transformer] is sufficient.

The configuration input can be found in [5, TPS SystemTemplate] and [6, TPS-SoftwareComponentTemplate].

[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 [1, ASWS Transformer] and defines an 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 Receiver ComSpec 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.disable EndToEndCheck is used.

[SWS_E2EXf_00134] ⌈The configuration options in EndToEndTransformationCom SpecProps 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 EndToEndTransformation ISignalProps.

[SWS_E2EXf_00154] ⌈If configuration option EndToEndTransformationComSpec Props.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>= emtpy (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 [5, System Template], [6, Software Component Template] and the generated ECU configuration (described in [1, General Specification of Transformers]), 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_PXXCheck StateType, 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 PXX.⌋*(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, E2E_SMConfigType and (if required) additional information (e.g. the Source ID on client side for profiles P04m, P07m, P08m, P44m) to perform E2E Protection within the E2E Transformer, from:

1. the number of E2E-protected data uniquely identified with <transformerId>, protected by profile PXX, 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_00177] ⌈The generated configuration object of type E2E_P08ConfigType 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_P11Config Type 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_P22Config Type 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_00178] ⌈The generated configuration object of type E2E_P04mConfig Type 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_00179] ⌈The generated configuration object of type E2E_P07mConfig Type 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_00204] ⌈The generated configuration object of type E2E_P08mConfig Type 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_00205] ⌈The generated configuration object of type E2E_P44mConfig Type 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_00176] ⌈The generated configuration object of type E2E_P44Config Type 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_00088] ⌈The generated config structure of type E2E_SMConfig Type,shall be initialized according to the following (one-to-one mapping):

- WindowSize = EndToEndTransformationComSpecProps.windowSize or EndTo EndTransformationDescription.windowSize
- MinOkStateInit = EndToEndTransformationComSpecProps.minOkStateInit or EndToEndTransformationDescription.minOkStateInit
- MaxErrorStateInit = EndToEndTransformationComSpecProps.maxErrorStateInit or EndToEndTransformationDescription.maxErrorStateInit
- MinOkStateValid = EndToEndTransformationComSpecProps.minOkStateValid or EndToEndTransformationDescription.minOkStateValid
- MaxErrorStateValid = EndToEndTransformationComSpecProps.maxErrorState Valid or EndToEndTransformationDescription.maxErrorStateValid
- MinOkStateInvalid = EndToEndTransformationComSpecProps.minOkStateInvalid or EndToEndTransformationDescription.minOkStateInvalid
- MaxErrorStateInvalid = EndToEndTransformationComSpecProps.maxErrorState Invalid 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*)

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 this is the case, 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 the following 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 the internal state data resources of E2E functions forward, protect check and state machine of E2E transformer.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00159] ⌈In case of post-build configuration, E2EXf_Init() function shall initialize the E2E transformer for a driving cycle.⌋*(RS_E2E_08538)*

[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 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)

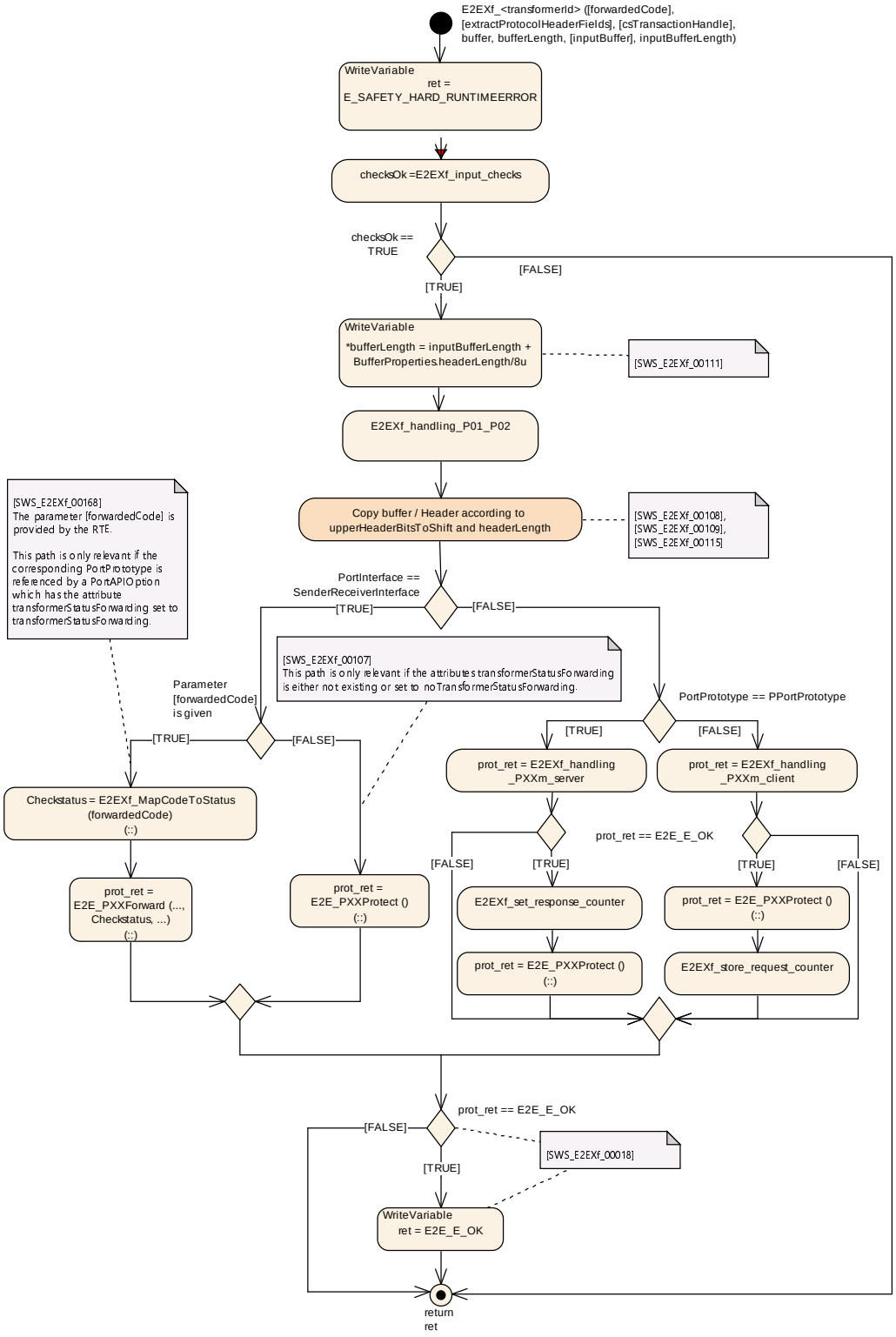

Figure 7.1: E2EXf_<transformerId>function overview

The 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 sent E2E-protected data element and each protected C/S operation (<transformerId>).⌋*(RS_E2E_08538)*

7.6.2.1 E2EXf_input_checks

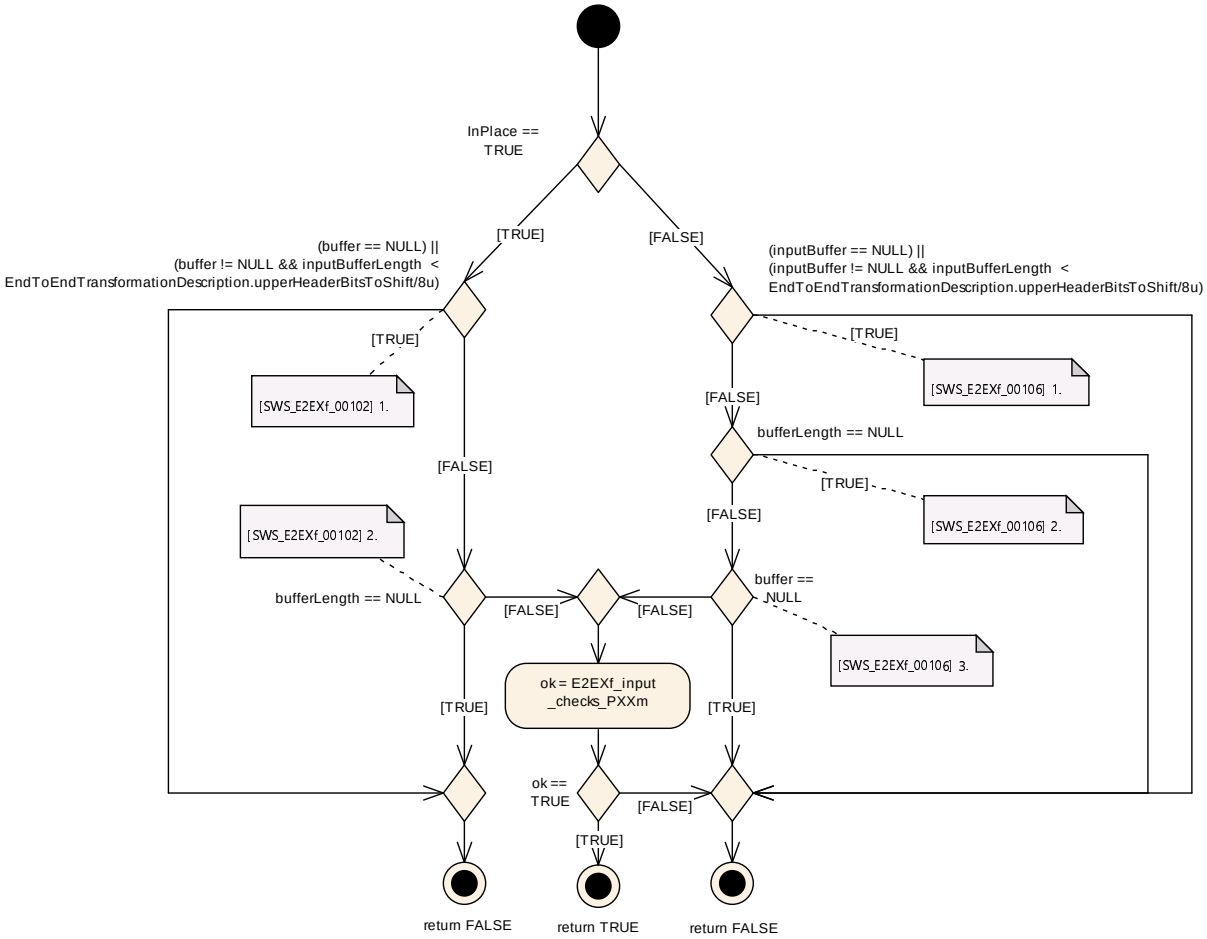

Figure 7.2: E2EXf_input_checks()

[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)*

[SWS_E2EXf_00180] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_<transformerId> shall perform the following additional precondition checks, without continuing further processing:

1. extractProtocolHeaderFields == NULL

2. csTransactionHandle == NULL.

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

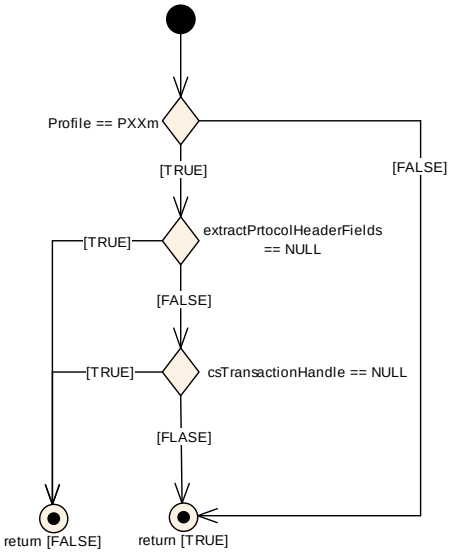

Figure 7.3: E2EXf_input_checks_PXXm()

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.

7.6.2.2 Copy Buffer/Header

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

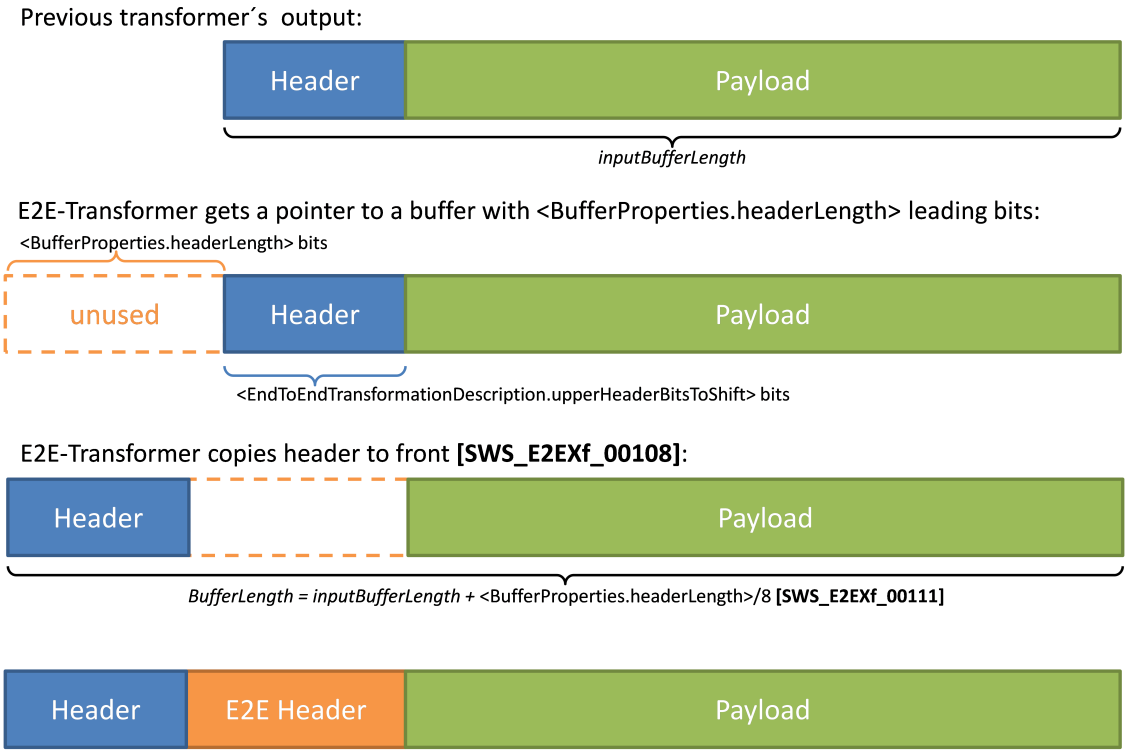

Figure 7.4: Buffer Handling of E2E_Transformer_<transformerID>

Figure 7.4 illustrates the buffer handling done by API function E2EXf_<transformer Id>for In-Place.

[SWS_E2EXf_00109] ⌈If (EndToEndTransformationDescription.upperHeaderBitsToShift > 0), out-of-place E2EXf_<transformerId> shall copy the first upper HeaderBits ToShiftbits 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)*

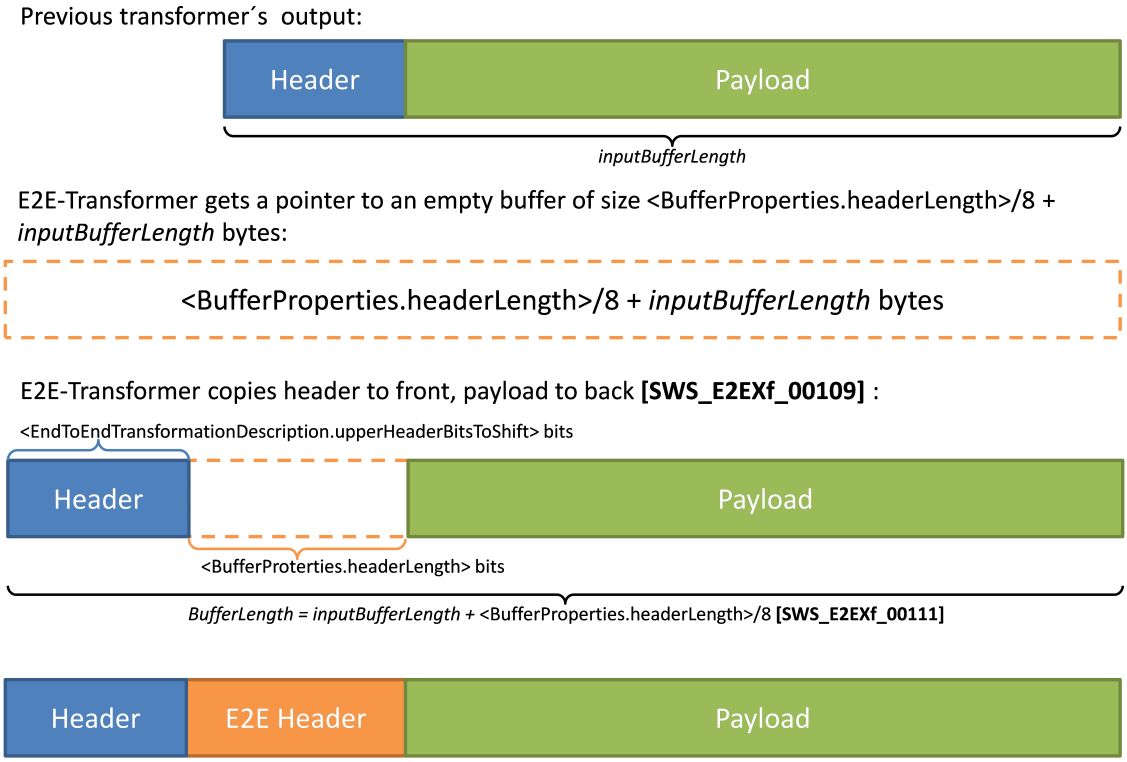

Figure 7.5: Header Shift Out of Place

Figure 7.5 illustrates the buffer handling done by API function E2EXf_<transformer Id>for Out-of-place.

[SWS_E2EXf_00111] ⌈E2EXf_<transformerId> shall set *bufferLength = inputBuffer Length + BufferProperties.headerLength/8.⌋ *(RS_E2E_08538)*

7.6.2.3 E2EXf_handling_P01_P02

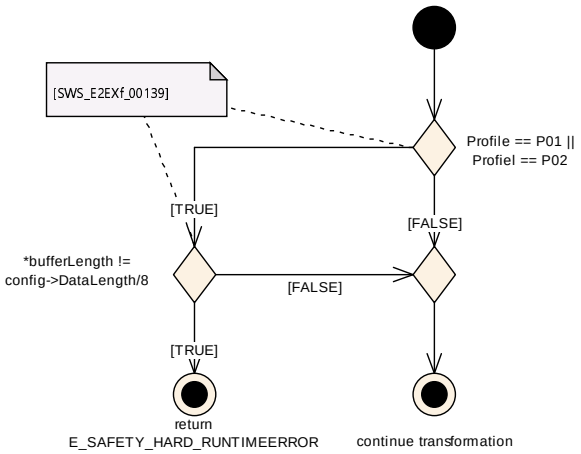

Figure 7.6: E2EXf_handling_P01_P02

[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 E_SAFETY_HARD_RUNTIMEERROR, i.e. without calling an E2E Library function.⌋*(RS_E2E_08538)*

Explicit length checks are only required for profiles 01 and 02, since the remaining profiles contain a input validation function. Because of this there is no need to check if the calculation of bufferLength did overflow.

In case SOME/IP based transformer is used, the E2E header of profile 1, 2, 11 and 22 are extended by 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]{DRAFT} ⌈If (((PXX = 01 &&dataIDMode != nibble) || (PXX == 02) || (PXX = 11 &&dataIDMode != nibble) || (PXX == 22)) && BufferProperties.header Length == 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)*

7.6.2.4 E2EXf_store_request_counter

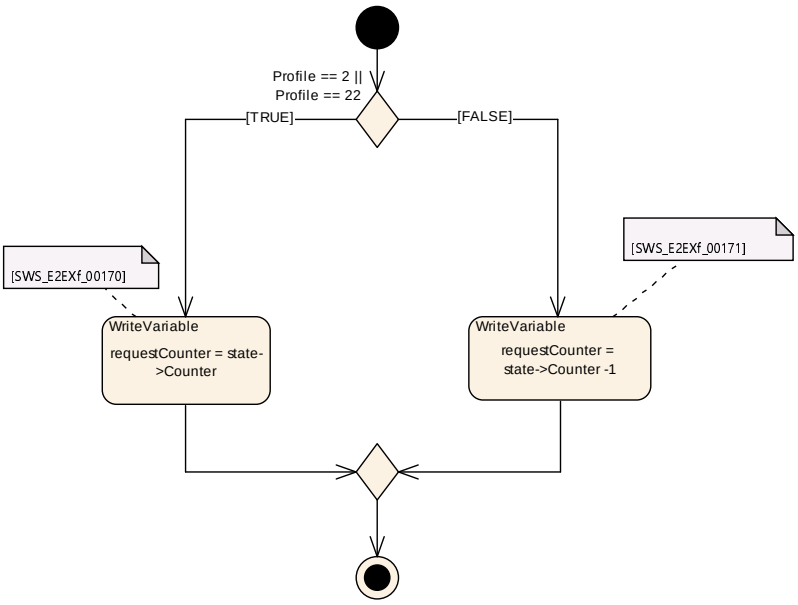

Figure 7.7: E2EXf_store_request_counter

When the Client receives the response of the server, the counter of the message shall be the same as the counter of the request. Therefore, requestCounter is used as an input to the check function.

[SWS_E2EXf_00170] ⌈If E2EXf_<transformerId> is used in a Client-Server Communication on the client-side and Profile is P02 or P22, state->Counter shall be stored to as requestCounter to be accessed by the E2EXf_Inv_<transformerId> for checking the response.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00171] ⌈If E2EXf_<transformerId> is used in a Client-Server Communication on the client-side and Profile is P01, P04, P05, P06, P07, P08, P11, P44, P04m, P07m, P08m, P44m state->Counter -1 shall be stored as requestCounter to be accessed by the E2EXf_Inv_<transformerId> for checking the response.⌋*(RS_E2E_08538)*

The Check function needs to access the counter value of the request in order to compare it to the counter value of the response from the server. If the counters do not match, an error occurred at the server.

7.6.2.5 E2EXf_handling_PXXm

[SWS_E2EXf_00181] ⌈For profiles P04m, P07m, P08m and P44m the in-place E2EXf_<transformerId> on the client-side shall call the extractProtocolHeaderFields() function passing the buffer, the bufferLength, the address of local messageType variable, and the address of a local messageResult variable as parameters.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00182] ⌈For profiles P04m, P07m, P08m and P44m the out-of-place E2EXf_<transformerId> on the client-side shall call the extractProtocolHeaderFields() function passing the inputBuffer, the inputBufferLength, the address of a local variable named messageType, and the address of a local variable named messageResult as parameters.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00183] ⌈If extractProtocolHeaderFields() returns something different from E_OK, E2EXf_<transformerId> shall return E_SAFETY_HARD_RUNTIMEERROR.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00184] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_<transformerId> on the client-side shall set a local variable sourceID to the sourceID stored in the configuration (see SWS_E2EXf_00126).⌋*(RS_E2E_08538)*

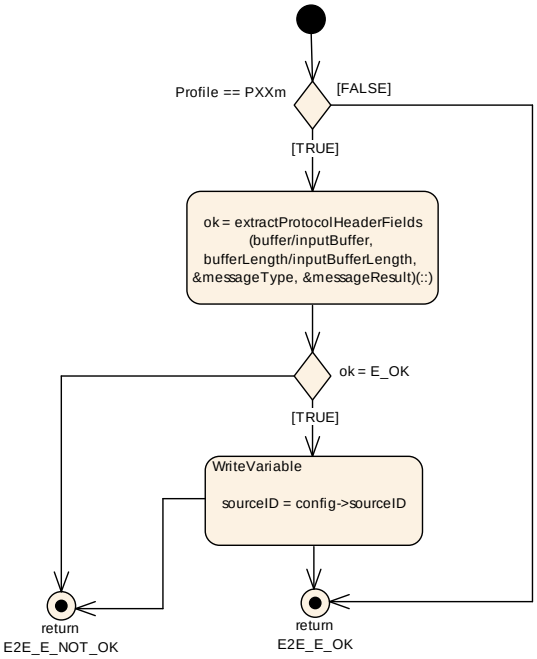

Figure 7.8: E2EXf_handling_PXXm_client

[SWS_E2EXf_00185] ⌈For profiles P04m, P07m, P08m and P44m the in-place E2EXf_<transformerId> on the server-side shall call the extractProtocolHeaderFields() function passing the buffer, the bufferLength, the address of local messageType variable, and the address of a local messageResult variable as parameters.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00186] ⌈For profiles P04m, P07m, P08m and P44m the out-of-place E2EXf_<transformerId> on the server-side shall call the extractProtocolHeaderFields() function passing the inputBuffer, the inputBufferLength, the address of local message Type variable, and the address of a local messageResult variable as parameters.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00187] ⌈If extractProtocolHeaderFields() returns something different from E_OK, E2EXf_<transformerId> shall return E_SAFETY_HARD_RUNTIMEERROR.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00188] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_<transformerId> on the server-side shall set a local variable sourceID to the value of the e2eSourceId element of the csTransactionHandle.⌋ *(RS_E2E_08538)*

[SWS_E2EXf_00189] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_<transformerId> on the server-side shall set a local variable named requestCounter to the value of the e2eCounter element of the csTransactionHandle.⌋*(RS_E2E_08538)*

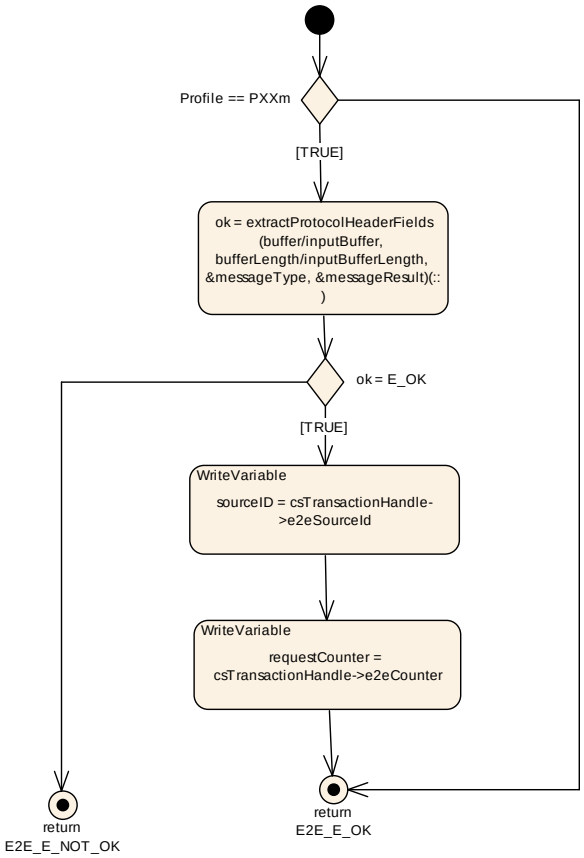

Figure 7.9: E2EXf_handling_PXXm_server

7.6.2.6 E2EXf_set_response_counter

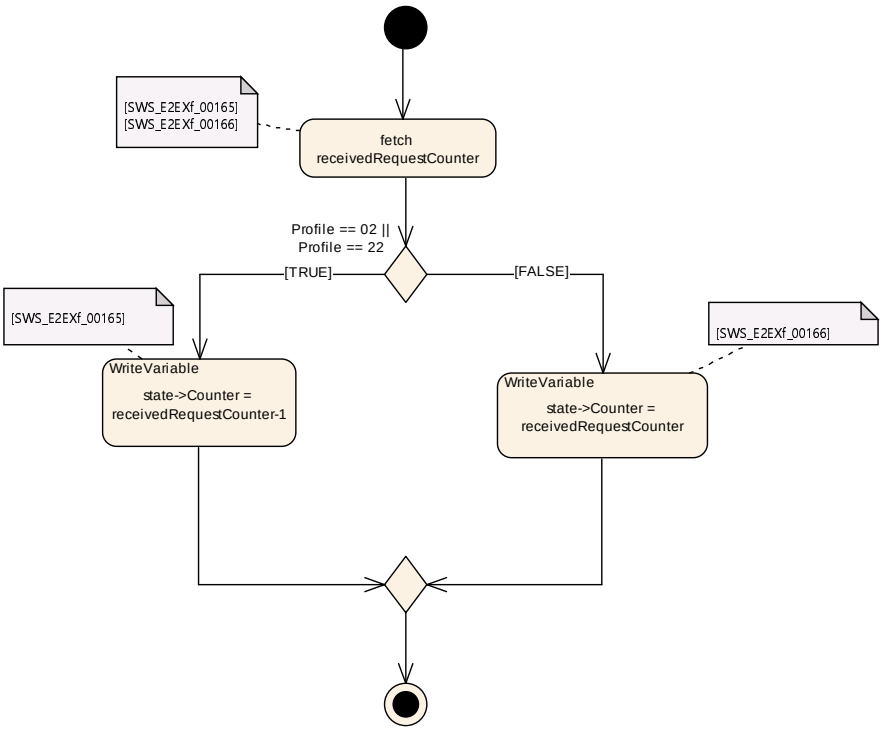

Figure 7.10: E2EXf_set_response_counter

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

Note: Using receivedRequestCounter-1 for Profiles P02 and P22 is motivated by the fact that the Protect() and Forward() functions of this profile increment the counter prior to adding it to the message/including it in the CRC calculation.

[SWS_E2EXf_00166] ⌈If E2E-Transformer is used in a Client-Server Communication and Profile is P01, P04, P04m, P05, P06, P07, P07m, P08, P08m, P11, P44 or P44m the sequence counter for protecting the response shall be set to requestCounter.⌋ *(RS_E2E_08538)*

Implementation Hint for profiles different from P04m, P07m, P08m and P44m Server-side: Access E2E_PXXCheckStateType* State, of the corresponding E2EXf_Inv inside the E2EXf call to get the last requestCounter for this Client-Server-Communication, this only works under the assumption connected Request-Response of a Client-Server-Call are handled synchronously.

Implementation Hint Client-side: The protect functions counter can be read out by the E2EXf_Inv by accessing the E2E_PXXProtectStateType* State of the E2EXf. This only works if the Client sends a Client/Server request and waits for the response of the server. If not, some counter buffer would be required. In case of Profile P01, P04, P05, P06, P07, P08, P11, P44, P04m, P07m, P08m or P44m : the counter of the protect state-1 contains the sent counter of the message.

7.6.2.7 E2EXf_MapCodeToStatus

The function E2EXf_MapCodeToStatus() maps the forwardedCode of type Std_TransformerForwardCode from Rte_Write (SWS_Rte_01071), Rte_Send (SWS_Rte_01072), Rte_IWrite (SWS_Rte_03744) and Rte_IWriteRef (SWS_Rte_05509) to a generic check status of type E2E_PCheckStatusType, which can be used by E2E_PXXForward() function.

[SWS_E2EXf_00208] ⌈The function E2EXf_MapCodeToStatus shall return the values depending on the value of forwardedCode:⌋*(RS_E2E_08538)*

forwardedCode	CheckStatus
E_OK	E2E_P_OK
E_SAFETY_INVALID_REP	E2E_P_REPEATED
E_SAFETY_INVALID_SEQ	E2E_P_WRONGSEQUENCE
E_SAFETY_INVALID_CRC	E2E_P_ERROR

Table 7.1: Check Status Mapping of forwardedCode

7.6.2.8 E2EXf Data Transformation

[SWS_E2EXf_00107]{DRAFT} ⌈If DataTransformationStatusForwarding is set to no TransformerStatusForwarding and PXX is P01 ,P02 ,P04 ,P05 ,P06 ,P07 ,P08 ,P11 ,P22 or P44, 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, P08 ,P11 ,P22 and P44) that were updated in above requirements SWS_E2EXf_00108, SWS_E2EXf_00109, SWS_E2EXf_00115, SWS_E2EXf_00111.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00190] ⌈If DataTransformationStatusForwarding is set to noTransformerStatusForwarding and PXX = P04m, P07m, P08m or P44m the function E2EXf_<transformerId>() shall invoke E2E_PXXProtect(), passing to that function:

- the appropriate Config structure (see [SWS_E2EXf_00125]),
- the appropriate State structure (see [SWS_E2EXf_00126]),
- the local variables sourceID, messageType, and messageResult
- buffer + EndToEndTransformationDescription.upperHeaderBitsToShift
- bufferLength - EndToEndTransformationDescription.upperHeaderBitsToShift

Hereby buffer and bufferLength were updated according to the above requirements SWS_E2EXf_00108, SWS_E2EXf_00109, SWS_E2EXf_00115, SWS_E2EXf_00111.⌋*(RS_E2E_08538)*

Note: Modifying the start and the length of the buffer here causes the upper header (e.g., the header added by the SOME/IP transformer) to be omitted from E2E_PXXProtect().

[SWS_E2EXf_00168]{DRAFT} ⌈If DataTransformationStatusForwarding is set to transformerStatusForwarding and PXX is P01, P02, P04, P05, P06, P07, P08, P11, P22 or P44 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, P08, P11, P22 and P44) that were updated in above requirements SWS_E2EXf_00108, SWS_E2EXf_00109, SWS_E2EXf_00115, SWS_E2EXf_00111.

In addition, the E2E status shall be passed on to the E2E_PXXForward() function based on the parameter forwardedCode provided by the RTE. This parameter is associated with the optional IN parameter forwardedCode from Rte_Write (SWS_Rte_01071), Rte_Send (SWS_Rte_01072), Rte_IWrite (SWS_Rte_03744) and Rte_IWriteRef (SWS_Rte_05509). The forwardedCode must be mapped to the matching E2E status (see Table 7.1).

⌋*(RS_E2E_08538)*

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

[SWS_E2EXf_00191] ⌈If DataTransformationStatusForwarding is set to transformer StatusForwarding and PXX = P04m, P07m, P08m or P44m: The function E2EXf_<transformerId>() shall invoke E2E_PXXForward(), passing to that function:

- the appropriate Config structure (see [SWS_E2EXf_00125]),
- the appropriate State structure (see [SWS_E2EXf_00126]),
- the local variables sourceID, messageType, and messageResult
- buffer + EndToEndTransformationDescription.upperHeaderBitsToShift
- bufferLength - EndToEndTransformationDescription.upperHeaderBitsToShift

Hereby buffer and bufferLength were updated according to the above requirements SWS_E2EXf_00108, SWS_E2EXf_00109, SWS_E2EXf_00115, SWS_E2EXf_00111.⌋*(RS_E2E_08538)*

Note: Modifying the start and the length of the buffer here causes the upper header (e.g., the header added by the SOME/IP transformer) to be omitted from E2E_PXXForward().

[SWS_E2EXf_00018]{DRAFT} ⌈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)

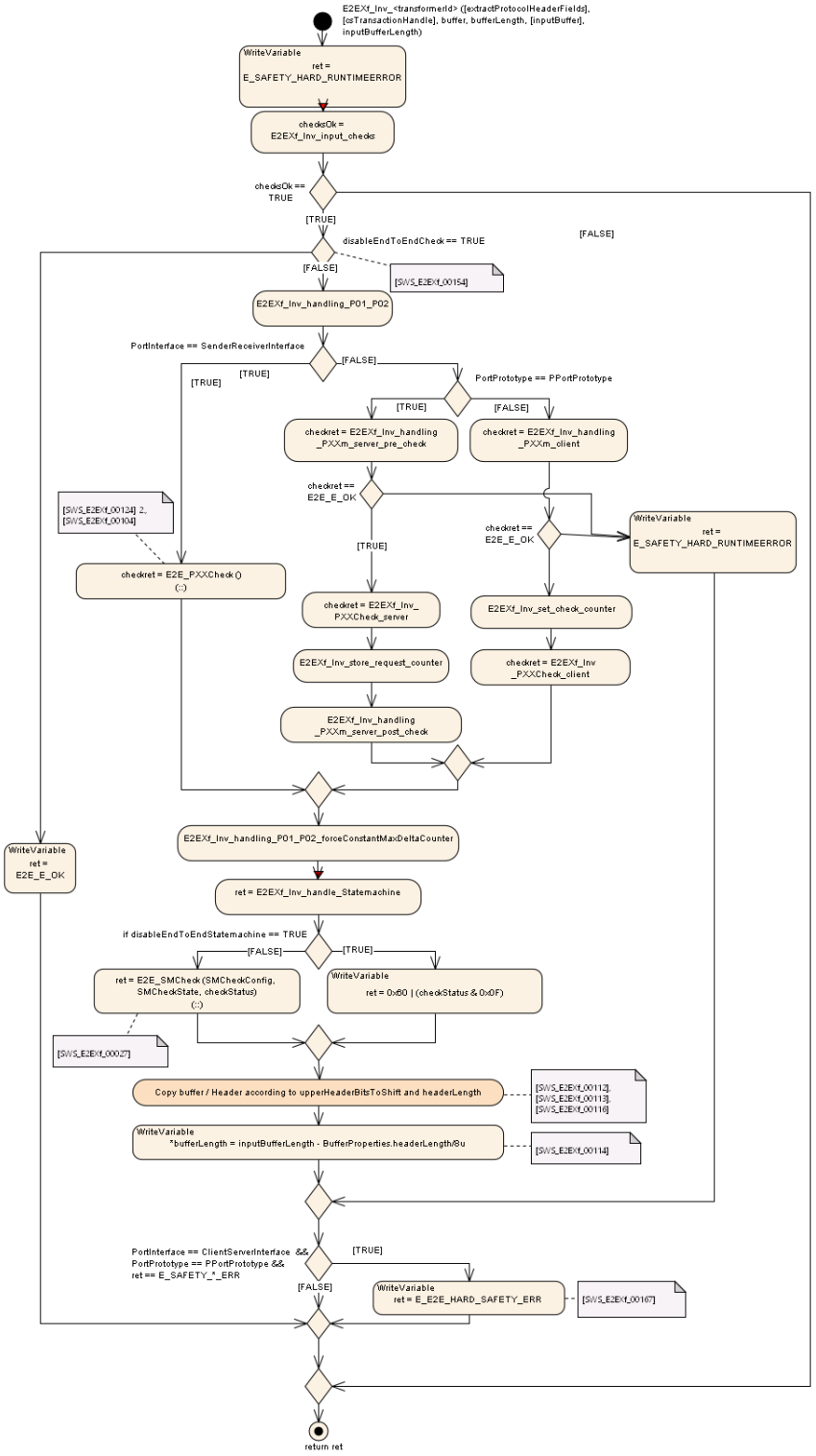

Figure 7.11: E2EXf_Inv_<transformerId>function overview

The 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 received E2E-protected data element and each protected C/S operation (<transformerId>).⌋ *(RS_E2E_08538)*

7.6.3.1 E2EXf_Inv_input_checks

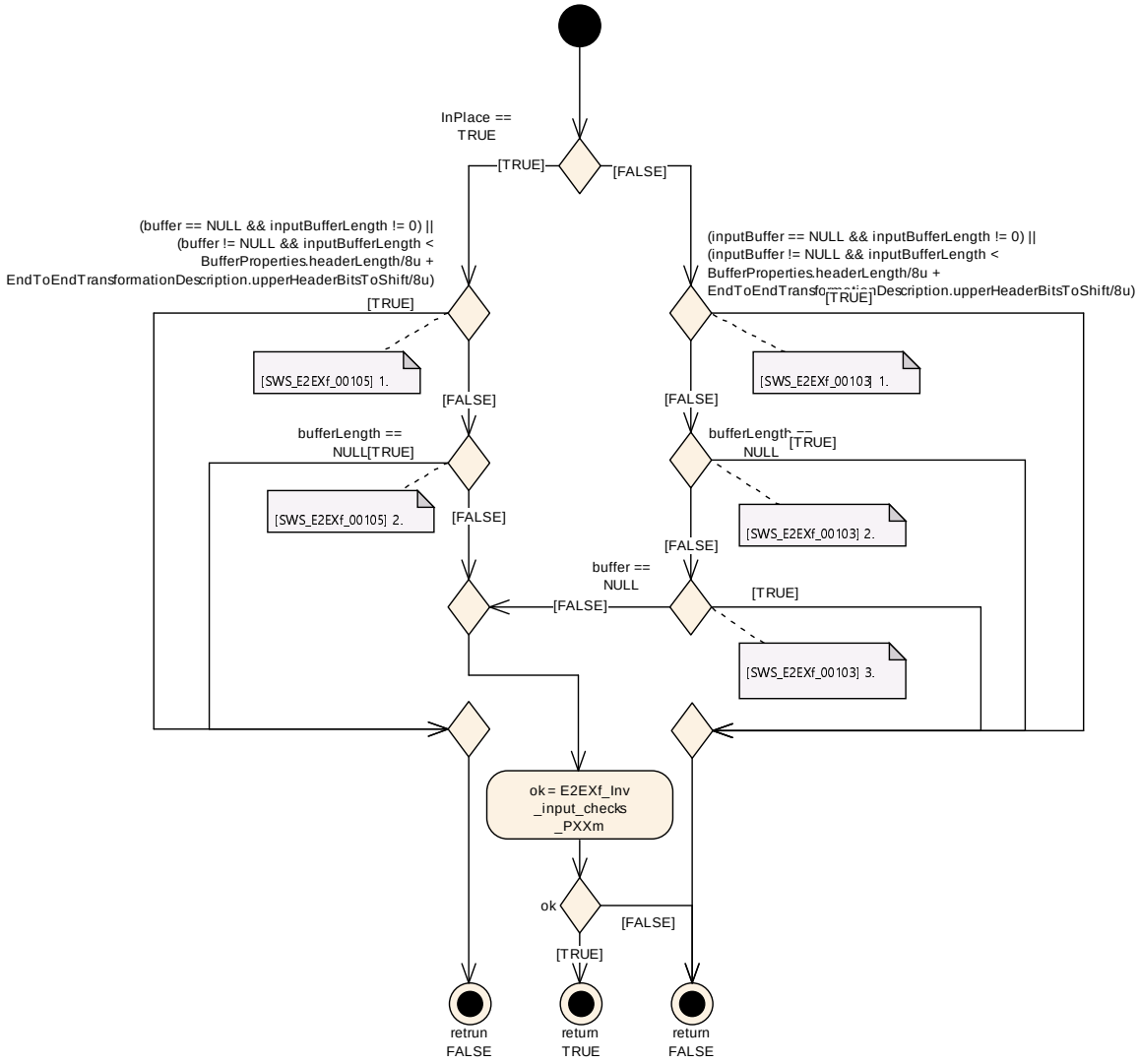

Figure 7.12: E2EXf_Inv_input_checks

[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)*

[SWS_E2EXf_00192] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_Inv_<transformerId> shall perform the following additional precondition checks, without continuing further processing:

1. extractProtocolHeaderFields == NULL

2. csTransactionHandle == NULL.

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

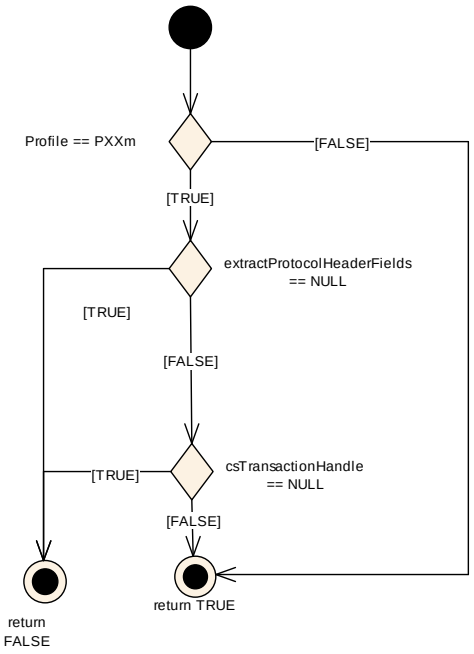

Figure 7.13: E2EXf_Inv_input_checks_PXXm

Depending on the configuration of the E2EXf regarding buffer operations (in-place and out-of-place), a valid pointer to the output buffer with length parameter too small, may result in a soft or hard error, depending on E2EXf configuration and actual given length. In this case the following values shall be returned:

Condition	Return Value	Description
Valid pointer to the output buffer, but the length of the buffer is too small to perform the buffer operations.	E_SAFETY_HARD_RUNTIMEERROR	cannot perform header copy operation, leaving buffer with invalid data according to configuration. Hard runtime error must be returned.
E2E profiles 1 or 2: Valid pointer to the output buffer and sufficient length for buffer operation, but the size of the buffer doesn't match the 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).
All E2E profiles except 1 or 2: Valid pointer to the output buffer and sufficient size for buffer operation, but the length is smaller then E2E profile configuration.	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.

Note that the function E2EXf_Inv_<transformerId> may be realized by a plain function, an inline function or a macro (implementation-specific).

[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.
3. 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*)

7.6.3.2 E2EXf_Inv_handling_P01_P02

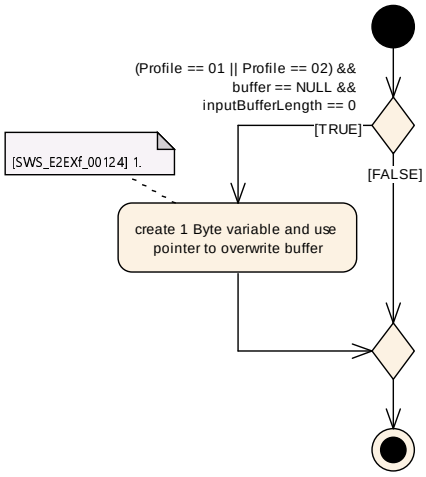

Figure 7.14: E2EXf_Inv_handling_P01_P02

[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)*

7.6.3.3 E2EXf_Inv_handling_P01_P02_forceConstantMaxDeltaCounter

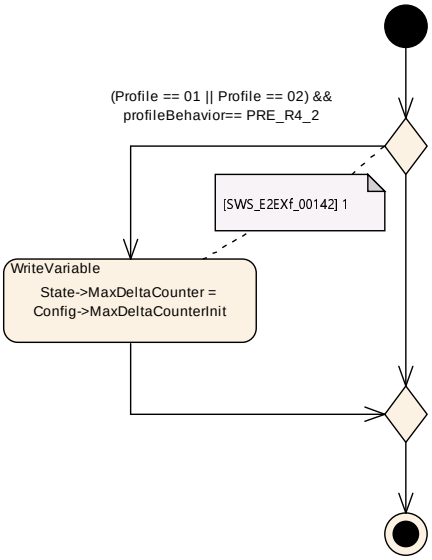

Figure 7.15: E2EXf_Inv_handling_P01_P02_forceConstantMxDeltaCounter

[SWS_E2EXf_00142] ⌈If configuration parameter profileBehavior is PRE_R4_2, then for PXX = 01 or 02, E2EXf_Inv_<transformerId> () shall set the variable MaxDelta Counter 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, 6, 7, 11 and 22. Profile 1 and 2 do not accept a NULL pointer, and they provide a sophisticated dynamic Max DeltaCounter and re-synchronization mechanism different to the less complex checks provided in the newer profiles.

However, changes in the legacy profiles 1 and 2 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 MaxNo NewOrRepeatedData 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)*

7.6.3.4 E2EXf_Inv_handling_PXXm

[SWS_E2EXf_00193] ⌈For profiles P04m, P07m, P08m and P44m the in-place E2EXf_Inv_<transformerId> on the client-side shall call the extractProtocolHeaderFields() function passing the buffer, the bufferLength, the address of local messageType variable, and the address of a local messageResult variable as parameters.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00194] ⌈For profiles P04m, P07m, P08m and P44m the out-of-place E2EXf_Inv_<transformerId> on the client-side shall call the extractProtocolHeaderFields() function passing the inputBuffer, the inputBufferLength, the address of a local variable named messageType, and the address of a local variable named messageResult as parameters.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00196] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_Inv_<transformerId> on the client-side shall set a local variable sourceID to the sourceID stored in the configuration (see SWS_E2EXf_00126).⌋*(RS_E2E_08538)*

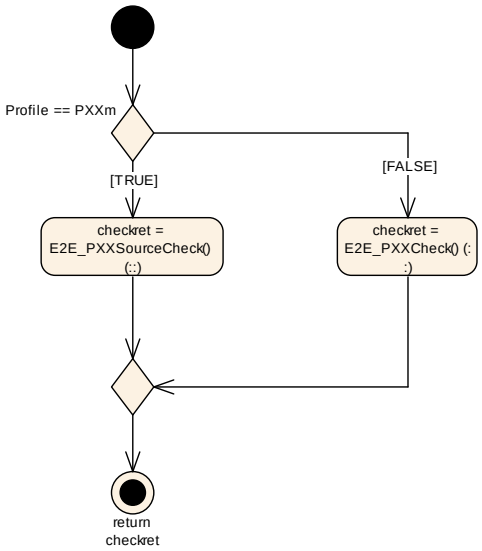

Figure 7.16: E2EXf_Inv_handling_PXXm_client

[SWS_E2EXf_00197] ⌈For profiles P04m, P07m, P08m and P44m the in-place E2EXf_Inv_<transformerId> on the server-side shall call the extractProtocolHeader Fields() function passing the buffer, the bufferLength, the address of local message Type variable, and the address of a local messageResult variable as parameters.⌋ *(RS_E2E_08538)*

[SWS_E2EXf_00198] ⌈For profiles P04m, P07m, P08m and P44m the out-of-place E2EXf_Inv_<transformerId> on the server-side shall call the extractProtocolHeader Fields() function passing the inputBuffer, the inputBufferLength, the address of local messageType variable, and the address of a local messageResult variable as parameters.⌋ *(RS_E2E_08538)*

[SWS_E2EXf_00199] ⌈If extractProtocolHeaderFields() returns something different from E_OK, E2EXf_Inv_<transformerId> shall return E_SAFETY_HARD_RUNTIMEERROR.⌋ *(RS_E2E_08538)*

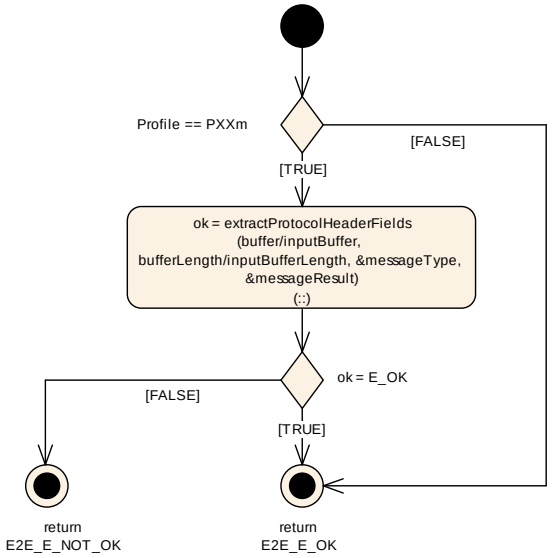

Figure 7.17: E2EXf_Inv_handling_PXXm_server_pre_check

[SWS_E2EXf_00200] ⌈For PXX = P04m, P07m, P08m and P44m: the function E2EXf_Inv_<transformerId> on the client-side shall invoke E2E_PXXSourceCheck(), passing to that function:

- config,
- state,
- the local variables messageType, messageResult, and the local variable source ID
- data length: inputBufferLength - EndToEndTransformationDescription.upperHeaderBitsToShift
- pointer to data:

  - inputBuffer + EndToEndTransformationDescription.upperHeaderBitsToShift (out-of-place version) or
  - buffer + EndToEndTransformationDescription.upperHeaderBitsToShift (in-place version).

⌋(*RS_E2E_08538*)

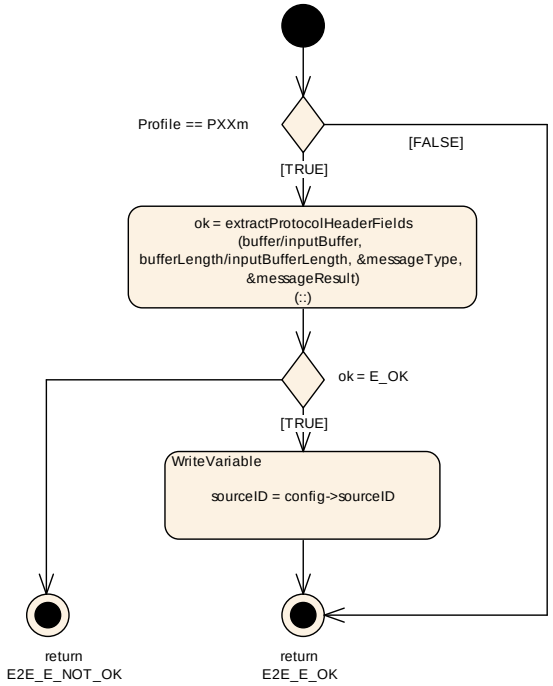

Figure 7.18: E2EXf_Inv_handling_PXXCheck_client

[SWS_E2EXf_00201] ⌈For PXX = P04m, P07m, P08m and P44m: the function E2EXf_Inv_<transformerId> on the server-side shall invoke E2E_PXXSinkCheck(), passing to that function:

- config,
- state,
- the local variables messageType, messageResult, and the address of the local variable sourceID
- data length: inputBufferLength - EndToEndTransformationDescription.upperHeaderBitsToShift
- pointer to data:
  - inputBuffer + EndToEndTransformationDescription.upperHeaderBitsToShift (out-of-place version) or
  - buffer + EndToEndTransformationDescription.upperHeaderBitsToShift (in-place version).

⌋(*RS_E2E_08538*)

Note: Modifying the start and the length of the inputBuffer/buffer here causes the upper header (e.g., the header added by the SOME/IP transformer) to be omitted from E2E_PXXSource/SinkCheck().

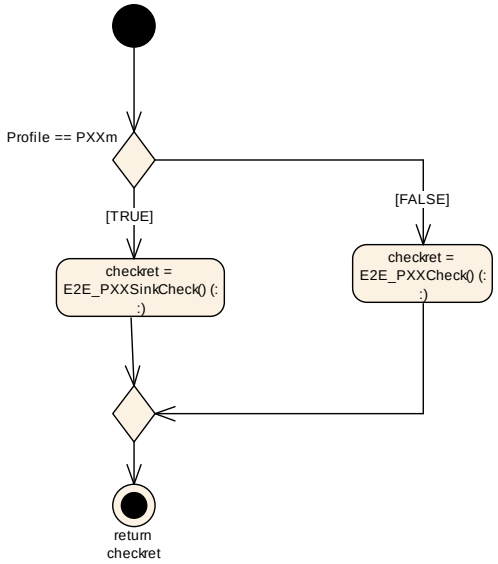

Figure 7.19: E2EXf_Inv_handling_PXXCheck_server

[SWS_E2EXf_00202] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_Inv_<transformerId> on the server-side shall set the e2e SourceId element of the csTransactionHandle to the value of the local variable source ID.⌋ *(RS_E2E_08538)*

[SWS_E2EXf_00203] ⌈For profiles P04m, P07m, P08m and P44m both the in-place and the out-of-place E2EXf_Inv_<transformerId> on the server-side shall set the e2e Counter element of the csTransactionHandle to the value of the local variable received RequestCounter.⌋ *(RS_E2E_08538)*

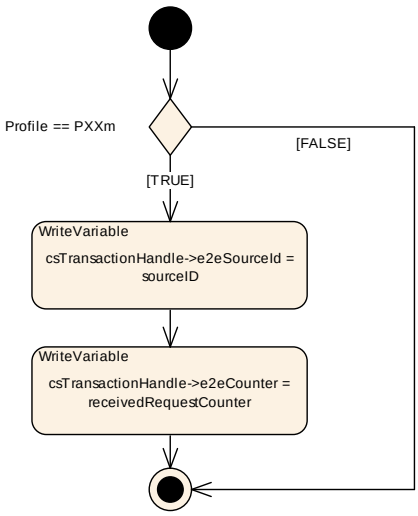

Figure 7.20: E2EXf_Inv_handling_PXXm_server_post_check

7.6.3.5 E2EXf_Inv_set_check_counter

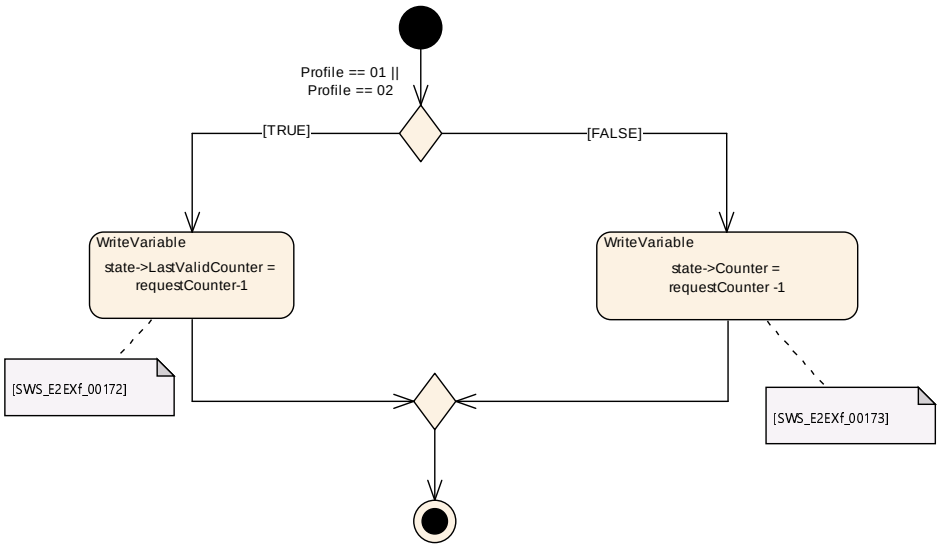

Figure 7.21: E2EXf_Inv_set_check_counter

[SWS_E2EXf_00172] ⌈In case of Client/Server Communication on the client side and Profile is P01 or P02, state->LastValidCounter of E2EXf_Inv_<transformer_Id> shall be set to the requestCounter-1.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00173] ⌈In case of Client/Server Communication on the client side and Profile is P04, P05, P06, P07, P08, P11, P22 or P44, P04m, P07m, P08m, P44m state->Counter of E2EXf_Inv_<transformerId> shall be set to the requestCounter-1.⌋ *(RS_E2E_08538)*

(RS_E2E_08538)

7.6.3.6 E2EXf_Inv_store_request_counter

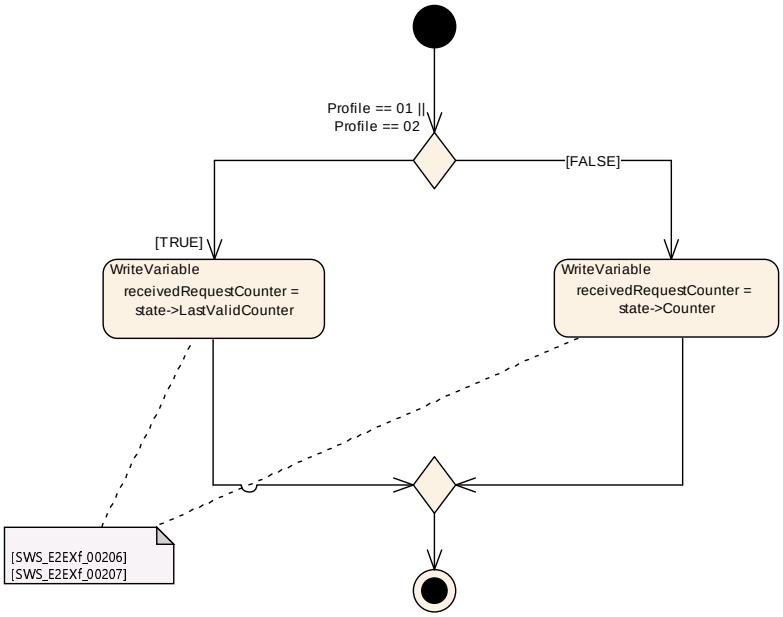

Figure 7.22: E2EXf_Inv_store_request_counter

[SWS_E2EXf_00206] ⌈In case of Client/Server Communication on the client side and Profile is P01 or P02, the receivedRequestCounter shall be set to state->LastValidCounter of E2EXf_Inv_<transformerId>.⌋*(RS_E2E_08538)*

[SWS_E2EXf_00207] ⌈In case of Client/Server Communication on the client side and Profile is P04, P05, P06, P07, P08, P11, P22, P44, P04m, P07m, P08m or P44m the receivedRequestCounter shall be set to state->Counter of E2EXf_Inv_<transformerId>.⌋*(RS_E2E_08538)*

This is the counter part for requirements SWS_E2EXf_00165 and SWS_E2EXf_00166.

7.6.3.7 E2EXf_Inv_handle_Statemachine

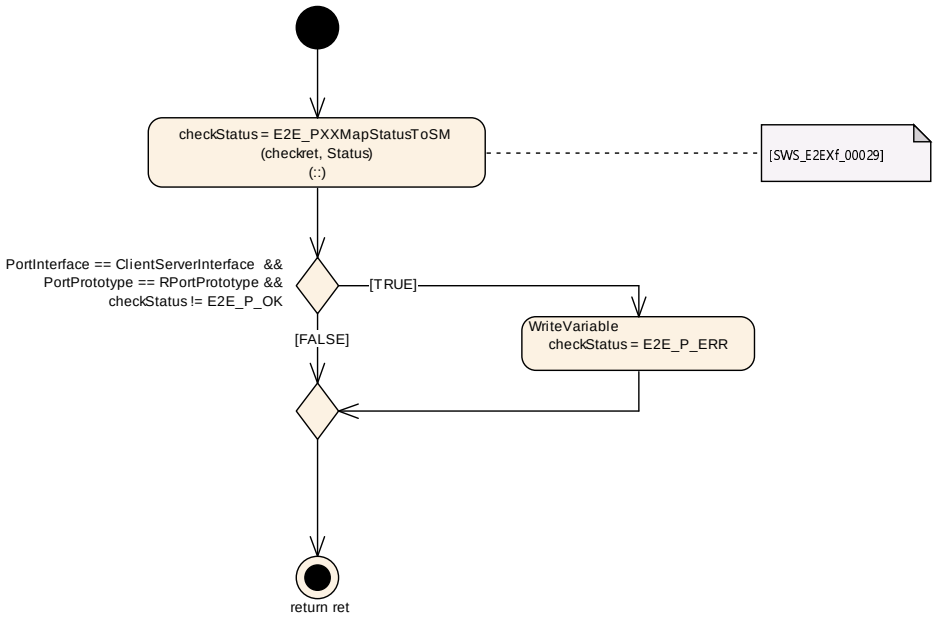

Figure 7.23: E2EXf_Inv_handle_Statemachine

[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_PXXCheck StateType, 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_00169] ⌈If disableEndToEndStateMachine is to TRUE,

- The high nibble of the return of the function E2EXf_Inv_<transformerId> shall be set to 0x6.
- 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_PCheck StatusType.

⌋(*RS_E2E_08538*)

[SWS_E2EXf_00027] ⌈If E2E_SMCheck() returns E2E_E_OK and disableEndToEnd StateMachine is FALSE, 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_PCheck StatusType.

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 upperHeader BitsToShift+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].

[SWS_E2EXf_00167] ⌈In case of Client/Server-Communication on the server side, if the return value ret equals to E_SAFETY_*_ERR, the value shall be overwritten to E_E2E_HARD_SAFETY_ERR .⌋(*RS_E2E_08538*)

The handling of E_SAFETY_*_ERR errors as hard errors creates a TransformerHard ErrorEvent. The intended use of this event is to inform the called SWC, that there was a failed Server-Client Call and possible some Error Handling has to be done. An error response can be created by adding a autonomous error reaction to this TransformerHardErrorEvent. The error response is then created by the RTE and the configured transformer chain without invoking the server application. The return value of

the transformer is then used as error code of the response. The client can determine if a E2E error occurred at the transmission of the request, by checking the return value. The chosen return value E_E2E_HARD_SAFETY_ERR aligns with the SOME/IP error code for a general E2E error, for more information see [7, SOME/IP Protocol Specification].

[SWS_E2EXf_00175] ⌈In case of Client/Server Communication on the client side, if the E2E_PXXCheck function returns a value different from E2E_P_OK, the status shall be set to E2E_P_ERR.⌋*(RS_E2E_08538)*

(RS_E2E_08538)

7.6.4 De-Initialization

[SWS_E2EXf_00148] ⌈E2EXf_DeInit() shall set the module initialization state to FALSE.⌋*(RS_E2E_08538)*

7.7 Error classification

7.7.1 Development Errors

The E2E Transformer shall be able to detect the following development errors

[SWS_E2EXf_00137] ⌈

Type of error	*Related error code*	*Error value*
Error code if any other API service, except Get VersionInfo is called before the transformer module was initialized with Init or after a call to De Init	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 this case 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 Trans-

former, 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. If not initialized, 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 this 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 and return.⌋ *(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, an the error shall be reported to the Default Error Tracer with either value E2EXF_E_PARAM (parameter out of range) resp. E2EXF_E_PARAM_POINTER in case of a pointer argument (NULL pointer) 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 error code E2EXF_E_UNINIT shall be reported to the Default Error Tracer 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 (NULL pointer) 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 [1, General Specification of Transformers].

Requirement SWS_Xfrm_00071 (XFRM_E_MALFORMED_MESSAGE) is not applicable for the E2E-Transformer.

8 API specification

8.1 Imported types

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

[SWS_E2EXf_00047] ⌈

Module	*Header File*	*Imported Type*
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_P04mCheckStateType
	E2E.h	E2E_P04mCheckStatusType
	E2E.h	E2E_P04mConfigType
	E2E.h	E2E_P04mProtectStateType
	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_P07mCheckStateType
	E2E.h	E2E_P07mCheckStatusType
	E2E.h	E2E_P07mConfigType
	E2E.h	E2E_P07mProtectStateType
	E2E.h	E2E_P08CheckStateType
	E2E.h	E2E_P08CheckStatusType

▽

△

Module	*Header File*	*Imported Type*
	E2E.h	E2E_P08ConfigType
	E2E.h	E2E_P08ProtectStateType
	E2E.h	E2E_P08mCheckStateType
	E2E.h	E2E_P08mCheckStatusType
	E2E.h	E2E_P08mConfigType
	E2E.h	E2E_P08mProtectStateType
	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_P44CheckStateType
	E2E.h	E2E_P44CheckStatusType
	E2E.h	E2E_P44ConfigType
	E2E.h	E2E_P44ProtectStateType
	E2E.h	E2E_P44mCheckStateType
	E2E.h	E2E_P44mCheckStatusType
	E2E.h	E2E_P44mConfigType
	E2E.h	E2E_P44mProtectStateType
	E2E.h	E2E_PCheckStatusType
	E2E.h	E2E_SMCheckStateType
	E2E.h	E2E_SMConfigType
	E2E.h	E2E_SMStateType
Std	Std_Types.h	Std_ExtractProtocolHeaderFieldsType
	Std_Types.h	Std_MessageResultType
	Std_Types.h	Std_MessageTypeType
	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_TransformerForwardCode (draft)
	Std_Types.h	Std_VersionInfoType

⌋() Furthermore, [1, ASWS Transformer] 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	

⌋(*RS_E2E_08538*)

8.2.2 E2EXf_CSTransactionHandleType

[SWS_E2EXf_00026] ⌈

Name	E2EXf_CSTransactionHandleType	
Kind	Structure	
Elements	e2eCounter	
	Type	uint32
	Comment	E2E Counter
	e2eSourceId	
	Type	uint32
	Comment	E2E Source ID
Description	Transaction handle for the C/S method call. - Used to tunnel the E2E Source ID and the E2E Counter from the request to the response at the server side via the RTE.	
Available via	E2EXf.h	

⌋(*RS_E2E_08538*)

8.3 Function definitions

8.3.1 E2EXf_<transformerId>

[SWS_E2EXf_00032] ⌈

Service Name	E2EXf_<transformerId>	

Syntax	`uint8 E2EXf_<transformerId> (`
	`  [Std_TransformerForwardCode forwardedCode],`
	`  [Std_ExtractProtocolHeaderFieldsType extractProtocolHeaderFields],`
	`  [const E2EXf_CSTransactionHandleType* csTransactionHandle],`
	`  uint8* buffer,`
	`  uint32* bufferLength,`
	`  [const uint8* inputBuffer],`
	`  uint32 inputBufferLength`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	forwardedCode	Optional parameter of E2E status forwarded to transformer chain and provided by the RTE. The presence of this parameter depends solely on the PortAPIOption transformerStatus Forwarding. The parameter must always be provided by the RTE if the corresponding PortPrototype is referenced by a Port APIOption which has the attribute transformerStatusForwarding set to transformerStatusForwarding.
	extractProtocolHeader Fields	Optional pointer to the function that shall be used to extract relevant protocol header fields of the serializing transformer in the transformer chain. Used for profiles PXXm.
	csTransactionHandle	Optional pointer to the E2E transaction handle for the C/S method call. - Used to tunnel the relevant information (E2E Source ID and E2E Counter) from the request to the response at the server side via the RTE. Used for profiles PXXm.
	inputBuffer	This argument only exists for E2E transformers configured for out-of-place transformation. It holds the input data for the transformer.
	inputBufferLength	This argument holds the length of the E2E transformer's input data (in the inputBuffer argument).
Parameters (inout)	buffer	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 headerLength different from 0, the output data of the previous transformer begin at position headerLength. 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)	bufferLength	Used length of the buffer

▽

△

Return value	uint8	0x00 (E_OK): Function performed successfully. 0xFF (E_SAFETY_HARD_RUNTIMEERROR): A runtime error occured, safety properties could not be checked and no output data could be produced.
Description	Protects the array/buffer to be transmitted, using the in-place transformation.	
Available via	E2EXf.h	

⌋(*RS_E2E_08538*) The return codes of E2EXf_<transformerId> are specified in TransformerTypes, see ASWS Transformer General.

8.3.2 E2EXf_Inv_<transformerId>

[SWS_E2EXf_00034] ⌈

Service Name	E2EXf_Inv_<transformerId>	

Syntax	`uint8 E2EXf_Inv_<transformerId> (`
	`  [Std_ExtractProtocolHeaderFieldsType extractProtocolHeaderFields],`
	`  [E2EXf_CSTransactionHandleType* csTransactionHandle],`
	`  uint8* buffer,`
	`  uint32* bufferLength,`
	`  [const uint8* inputBuffer],`
	`  uint32 inputBufferLength`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	extractProtocolHeader Fields	Optional pointer to the function that shall be used to extract relevant protocol header fields of the serializing transformer in the transformer chain. Used for profiles PXXm
	inputBuffer	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.
	inputBufferLength	This argument holds the length of the transformer's input data (in the inputBuffer argument). If executeDespiteDataUnavailability is set to true and the RTE cannot provide data as input to the transformer, the length will be equal to 0.
Parameters (inout)	buffer	This argument is only an INOUT argument for 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 executeDespiteData Unavailability is set to true and the RTE cannot provide data as input to the transformer, it will hand over a NULL pointer to the transformer. 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)	csTransactionHandle	Optional pointer to the E2E transaction handle for the C/S method call. - Used to tunnel the relevant information (E2E Source ID and E2E Counter) from the request to the response at the server side via the RTE. Used for profiles PXXm
	bufferLength	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: 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 occured, 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 occured, safety properties could not be checked and no output data could be produced. E_E2E_HARD_SAFETY_ERR
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	`void E2EXf_Init (`
	`  const E2EXf_ConfigType* config`
	`)`	

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	`void E2EXf_DeInit (`
	`  void`
	`)`

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	`void E2EXf_GetVersionInfo (`
	`  Std_VersionInfoType* versioninfo`
	`)`	

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

⌋*(RS_E2E_08538)*

8.4 Callback 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_P01Check	E2E.h	Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.
E2E_P01CheckInit	E2E.h	Initializes the check state
E2E_P01Forward (draft)	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_P01MapStatusToSM	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_P01Protect	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_P01ProtectInit	E2E.h	Initializes the protection state.
E2E_P02Check	E2E.h	Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.
E2E_P02CheckInit	E2E.h	Initializes the check state
E2E_P02Forward (draft)	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_P02MapStatusToSM	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_P02Protect	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_P02ProtectInit	E2E.h	Initializes the protection state.
E2E_P04Check	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_P04CheckInit	E2E.h	Initializes the check state
E2E_P04Forward (draft)	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_P04MapStatusToSM	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_P04mCheckInit	E2E.h	Initializes the check state

▽

△

API Function	Header File	Description
E2E_P04mForward (draft)	E2E.h	Protects data which is forwarded by using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft
E2E_P04mMapStatusToSM	E2E.h	The function maps the check status of Profile 4m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 4m delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P04mProtect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID.
E2E_P04mProtectInit	E2E.h	Initializes the protection state.
E2E_P04mSinkCheck	E2E.h	Checks the Data received using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).
E2E_P04mSourceCheck	E2E.h	Checks the Data received using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
E2E_P04Protect	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_P04ProtectInit	E2E.h	Initializes the protection state.
E2E_P05Check	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_P05CheckInit	E2E.h	Initializes the check state
E2E_P05Forward (draft)	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_P05MapStatusToSM	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.

▽

△

API Function	*Header File*	*Description*
E2E_P05Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 5. This includes checksum calculation, handling of counter.
E2E_P05ProtectInit	E2E.h	Initializes the protection state.
E2E_P06Check	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_P06CheckInit	E2E.h	Initializes the check state
E2E_P06Forward (draft)	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_P06MapStatusToSM	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_P06Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 6. This includes checksum calculation, handling of counter.
E2E_P06ProtectInit	E2E.h	Initializes the protection state.
E2E_P07Check	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_P07CheckInit	E2E.h	Initializes the check state
E2E_P07Forward (draft)	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_P07MapStatusToSM	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_P07mCheckInit	E2E.h	Initializes the check state
E2E_P07mForward (draft)	E2E.h	Protects data which is forwarded by using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft
E2E_P07mMapStatusToSM	E2E.h	The function maps the check status of Profile 7m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 7m delivers a more fine-granular status, but this is not relevant for the E2E state machine.

▽

△

API Function	*Header File*	*Description*
E2E_P07mProtect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID.
E2E_P07mProtectInit	E2E.h	Initializes the protection state.
E2E_P07mSinkCheck	E2E.h	Checks the Data received using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).
E2E_P07mSourceCheck	E2E.h	Checks the Data received using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
E2E_P07Protect	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_P07ProtectInit	E2E.h	Initializes the protection state.
E2E_P08Check	E2E.h	Checks the Data received using the E2E profile 08. 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_P08CheckInit	E2E.h	Initializes the check state
E2E_P08Forward (draft)	E2E.h	Protects data which is forwarded by using the E2E profile 08. 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_P08MapStatusToSM	E2E.h	The function maps the check status of Profile 08 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 08 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P08mCheckInit	E2E.h	Initializes the check state
E2E_P08mForward (draft)	E2E.h	Protects data which is forwarded by using the E2E profile 08m. 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_P08mMapStatusToSM	E2E.h	The function maps the check status of Profile 08m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 08m delivers a more fine-granular status, but this is not relevant for the E2E state machine.

▽

△

API Function	*Header File*	*Description*
E2E_P08mProtect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 08m. This includes CRC calculation, handling of counter, Data ID, Message Type, Message Result and Source ID.
E2E_P08mProtectInit	E2E.h	Initializes the protection state.
E2E_P08mSinkCheck	E2E.h	Checks the Data received using the E2E profile 8m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).
E2E_P08mSourceCheck	E2E.h	Checks the Data received using the E2E profile 8m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
E2E_P08Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 08. This includes checksum calculation, handling of counter and Data ID.
E2E_P08ProtectInit	E2E.h	Initializes the protection state.
E2E_P11Check	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_P11CheckInit	E2E.h	Initializes the check state
E2E_P11Forward (draft)	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_P11MapStatusToSM	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_P11Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 11. This includes checksum calculation, handling of counter.
E2E_P11ProtectInit	E2E.h	Initializes the protection state.
E2E_P22Check	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_P22CheckInit	E2E.h	Initializes the check state
E2E_P22Forward (draft)	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

▽

△

API Function	*Header File*	*Description*
E2E_P22MapStatusToSM	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_P22Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 22. This includes checksum calculation, handling of counter.
E2E_P22ProtectInit	E2E.h	Initializes the protection state.
E2E_P44Check	E2E.h	Checks the Data received using the E2E profile 44. 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_P44CheckInit	E2E.h	Initializes the check state.
E2E_P44Forward (draft)	E2E.h	Protects data which is forwarded by using the E2E profile 44. 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_P44MapStatusToSM	E2E.h	The function maps the check status of Profile 44 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 44 delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P44mCheckInit	E2E.h	Initializes the check state
E2E_P44mForward (draft)	E2E.h	Protects data which is forwarded by using the E2E profile 44m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft
E2E_P44mMapStatusToSM	E2E.h	The function maps the check status of Profile 44m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 44m delivers a more fine-granular status, but this is not relevant for the E2E state machine.
E2E_P44mProtectInit	E2E.h	Initializes the protection state.
E2E_P44mSinkCheck	E2E.h	Checks the Data received using the E2E profile 44m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).
E2E_P44mSourceCheck	E2E.h	Checks the Data received using the E2E profile 44m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
E2E_P44Protect	E2E.h	Protects the array/buffer to be transmitted using the E2E profile 44. This includes checksum calculation, handling of counter and Data ID.

▽

△

API Function	*Header File*	*Description*
E2E_P44ProtectInit	E2E.h	Initializes the protection state.
E2E_SMCheck	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_SMCheckInit	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>

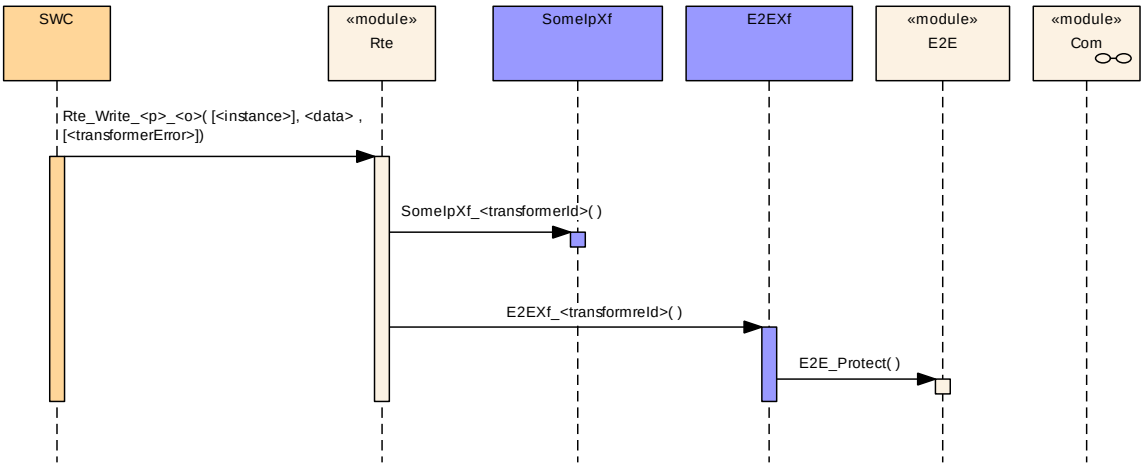

Figure 9.1: E2EXf_<TransformerID>

9.2 Check - E2EXf_Inv_<transformerId>

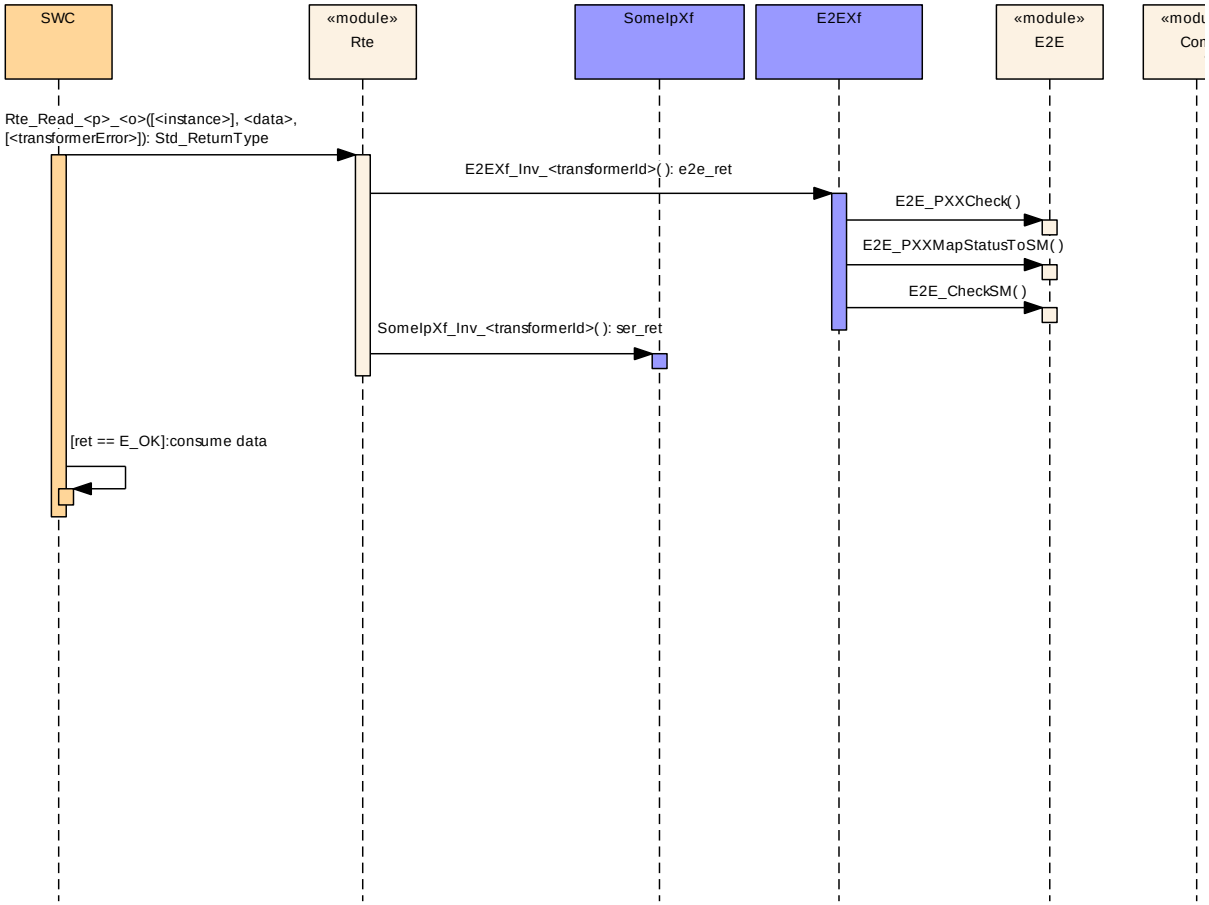

Figure 9.2: E2EXf_Inv_<TransformerID>

9.3 E2E For Methods

9.3.1 E2E Protected Method Call

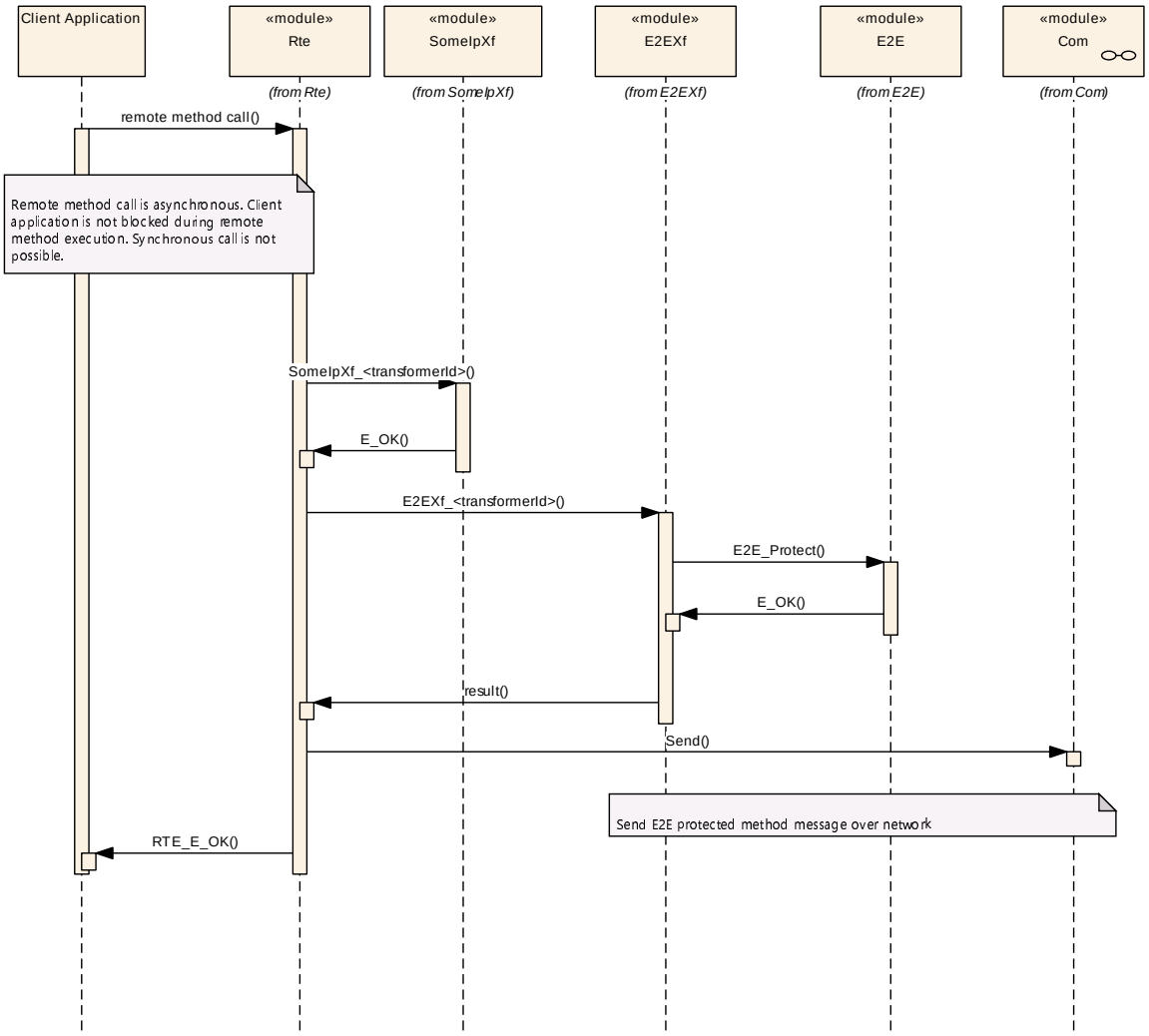

Figure 9.3: Send E2E Protected Method Call

The SWC calls a remote method which is executed somewhere else. This is an asynchronous call. The client application is not blocked when the remote method is executed at the remote side. The call by client application is first handled by RTE. This call is E2E protected before it is sent remote where it is executed. So RTE calls the Some Ip transformer for serializing and then the E2E transformer for adding E2E protection. The E2E protected message is sent over a network to its destination.

9.3.2 Receive E2E Protected Call

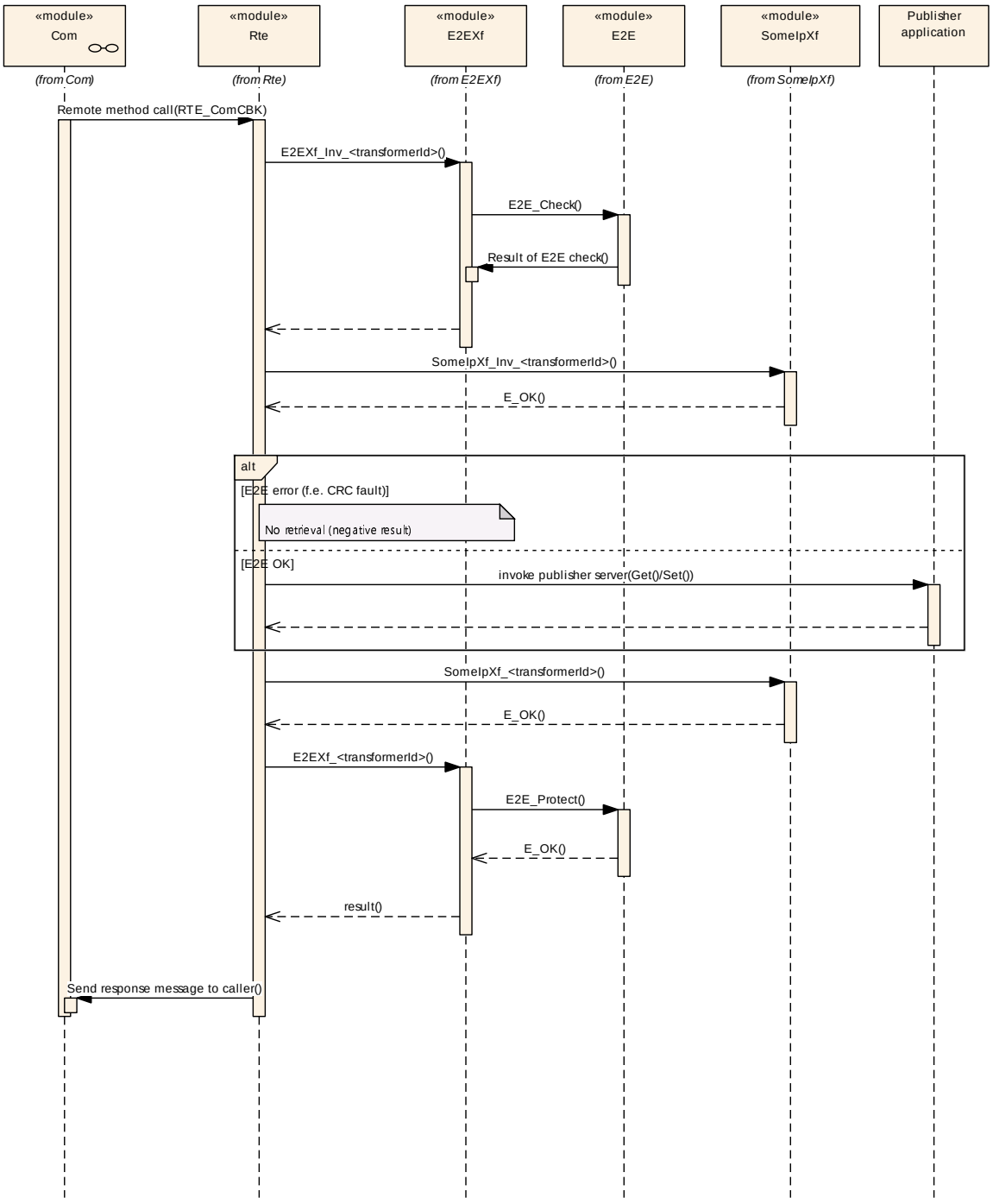

Figure 9.4: Receive E2E Protected Method Call

The E2E protected message comes in at COM and is forwarded to RTE. RTE calls E2E transformer to check for E2E errors and then - if there is not E2E error - SomeIp transformer for deserializing. The result of the E2E check decides if the remote function is called or not. In case of an E2E error there is no call.

In both cases (E2E error/no E2E error) a response message is set up and also serialized and E2E protected before it is sent back.

9.3.3 Response to an E2E Protected Call

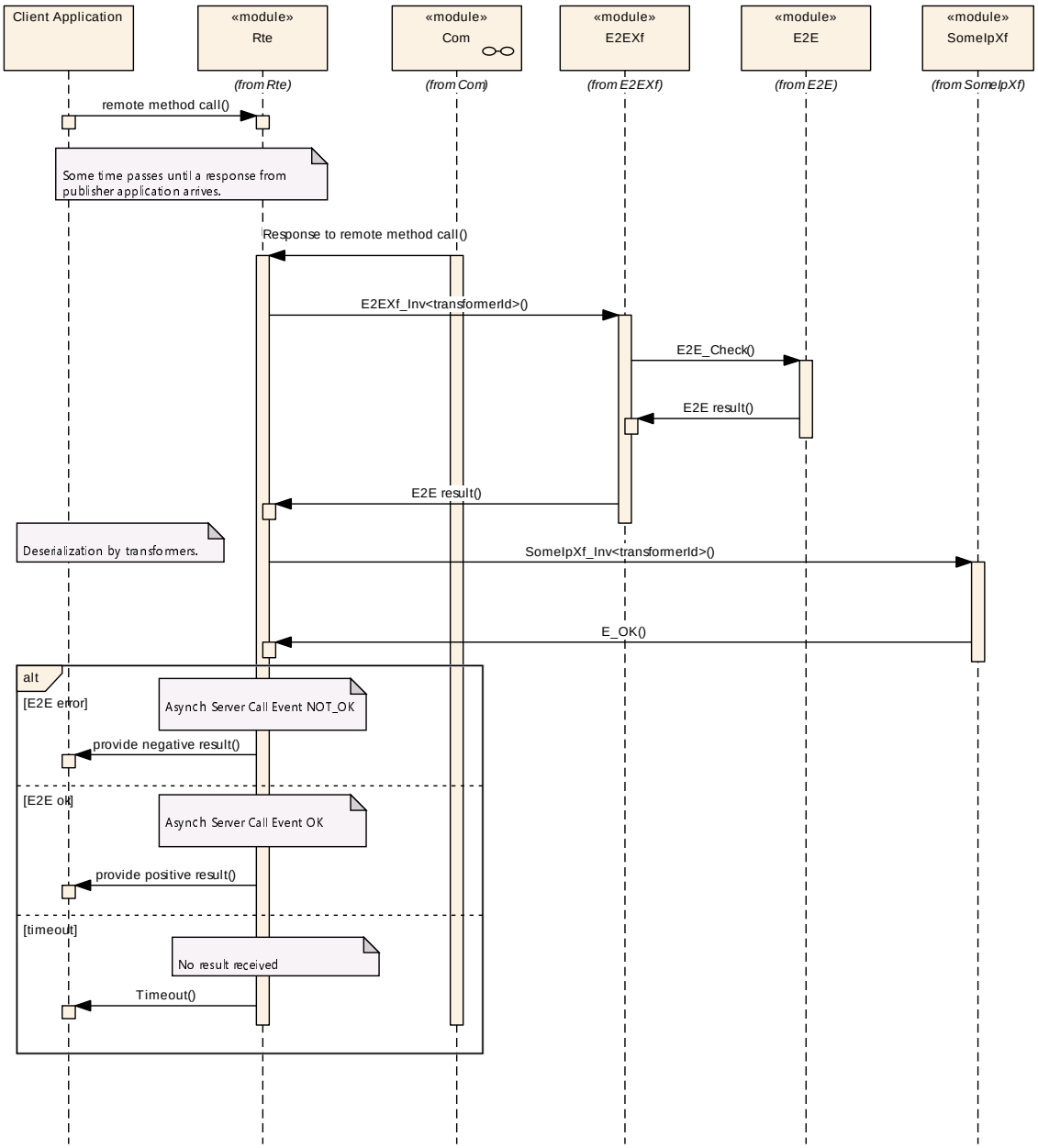

Figure 9.5: Response to an E2E Protected Method Call

The E2E protected response message comes in at COM and is forwarded to RTE. RTE calls E2E transformer and then SomeIp transformer to check for E2E errors. In case of an E2E error a negative result is provided to client application. So the client application is informed that no valid value is returned from remote. In case of no E2E error the client application can use the result values (positive result).

9.4 E2E For Events

9.4.1 Send an E2E Protected Event

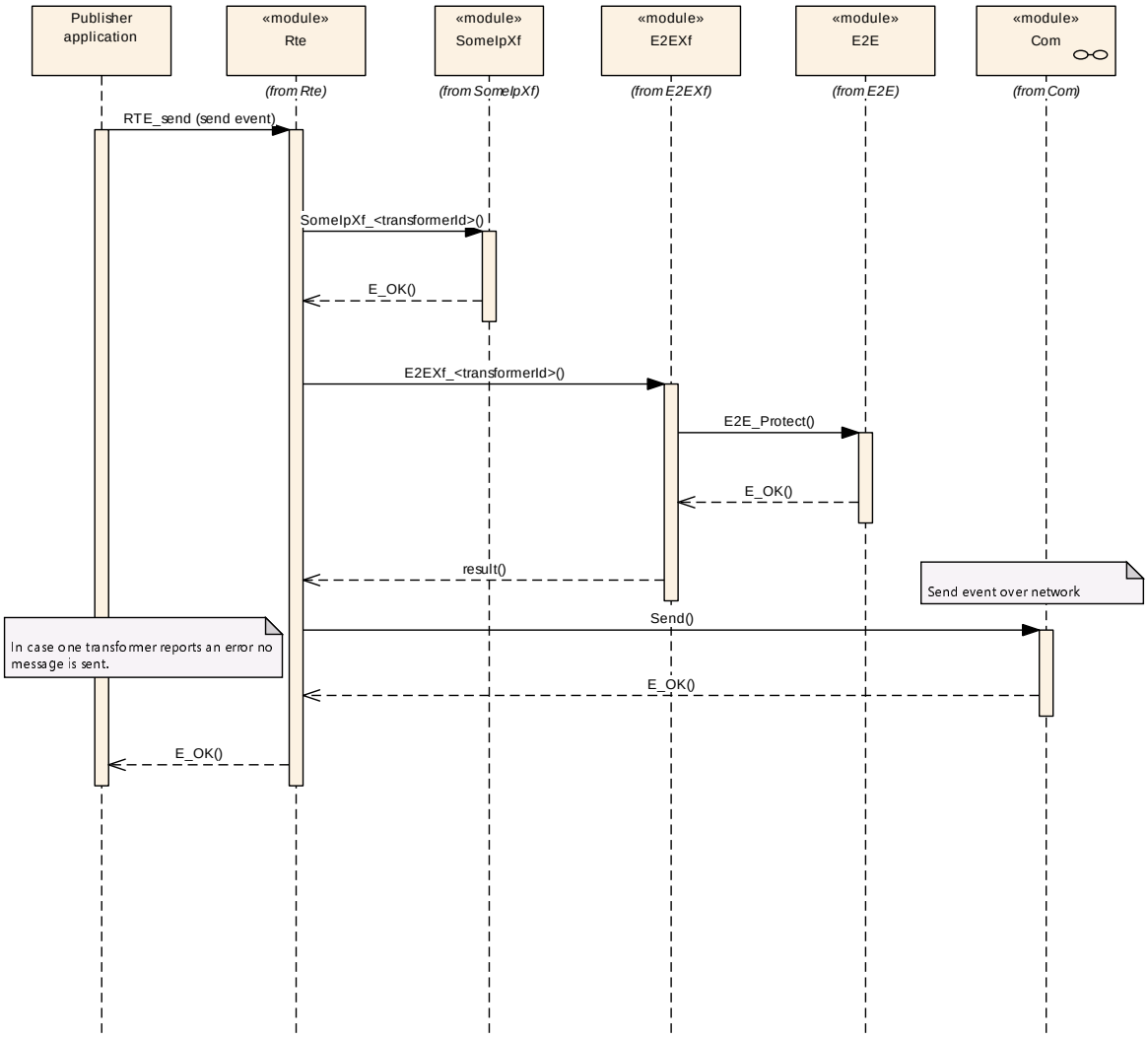

Figure 9.6: Send an E2E Protected Event

Here we have the publisher subscriber pattern. When there was an update of a field then all subscribers shall receive an update message. RTE calls SomeIp transformer (serialization) and E2E transformer for protection. The E2E protected event message is sent over network. This procedure is repeated for all subscribers to the field.

An event is be sent to all subscribers. This is done in a loop over all subscribers.

9.4.2 Receive an E2E Protected Event

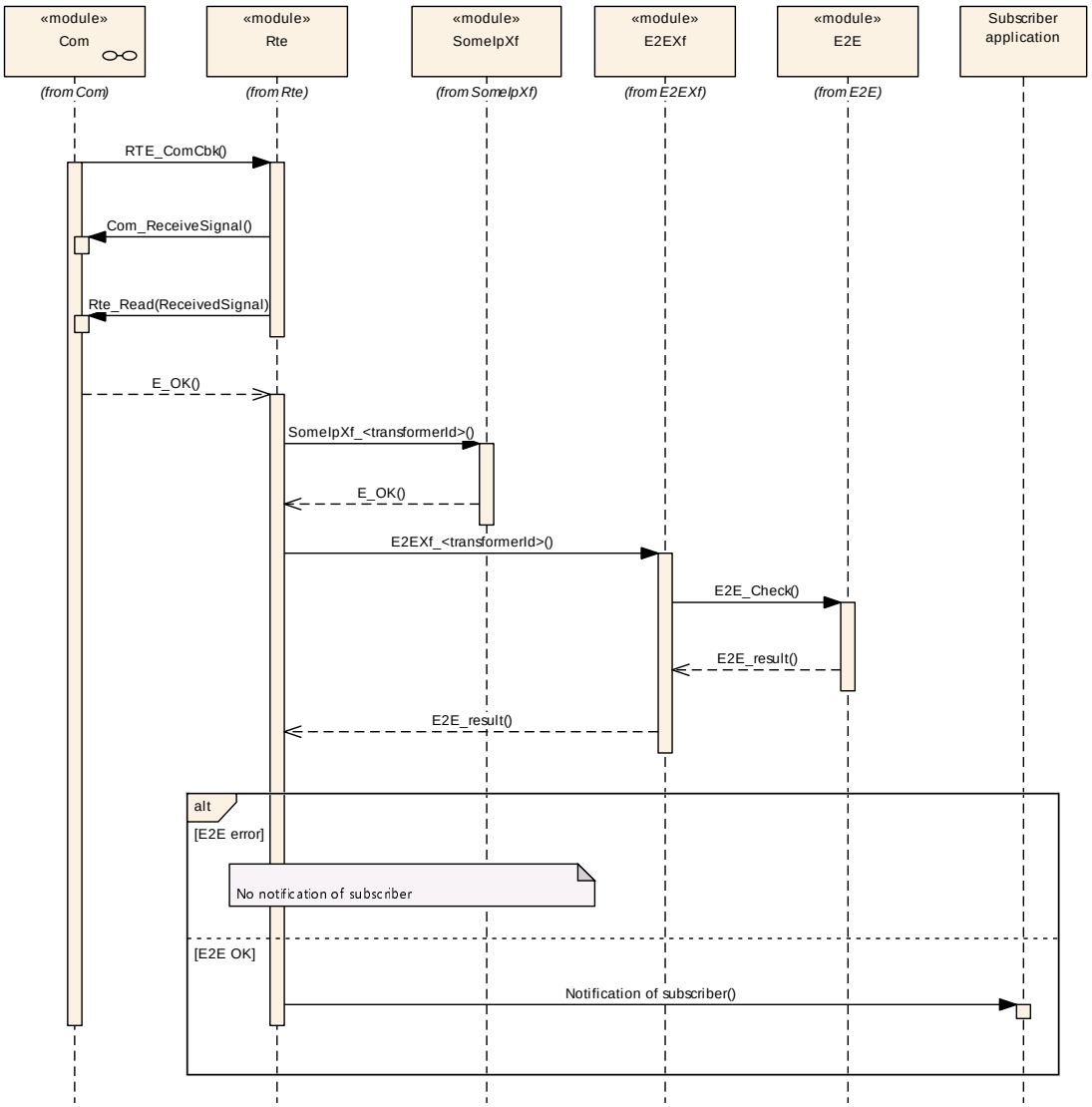

Figure 9.7: Receive an E2E Protected Event

The E2E protected event comes in at COM. RTE retrieves the message, calls E2E transformer to check if there was an E2E error and SomeIp Tranformer if there is no E2E error. In case of error the subscriber is not updated, in case of no E2E error the subscriber gets the updated value of the field.

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 the implementing BswModuleEntry entities).

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

A Not applicable requirements

[SWS_E2EXf_NA_00001] ⌈These requirements are not applicable to this specification.⌋*(SWS_Xfrm_00071)*