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	R24-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2024-11-27	R24-11	AUTOSAR Release Management	• No content changes
2023-11-23	R23-11	AUTOSAR Release Management	• Rework flowcharts, sequence charts and explaining text • Description of E2E state machine results updated
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.

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

E2E transformer belongs to the class "Safety", according to [1, SRS Transformer General] and is based on the specification in the [2, ASWS 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. If receiving SW-C does not evaluate the return codes of the E2E transformer, then the SW-C might use corrupted data.

The E2E transformer instantiates E2E configuration and E2E state data structures, based on its configuration. The E2E transformer is responsible for the invocation of the E2E Library based on the configuration of a specific data element (I-signal).

There is no data splitting/merging, i.e. one I-signal is NOT made of several data elements and one data element is not made of several I-signals. On the sender side, one data element maps exactly one-to-one to an I-signal. On the receiver side, one-or-more data elements represent the entire received I-signal. 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 is serialized to
   (a) **one I-signal** and protected with one E2E-protection
   (b) **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.
2. On receiver side, one I-signal checked
   (a) **Once**: resulting with one data element (i.e. no fan-out)
   (b) **Several times**: with the same settings, but by independent functions (e.g. by ASIL-independent receiver SW-Cs),
   (c) **Several times**: with partially different settings (e.g. same DataID, but different counter tolerances), by different functions, resulting with separate data elements each having possibly different E2E-check result,
   (d) **Several times**: with and without E2E-check enabled (e.g. if one receiver is safety-related, and another one is QM and it does not need the results of E2E check).

The E2E transformer is invoked by a SW-C via the RTE API (read, write, send, receive).

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Communication Management that are not included in the [3, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
I-Signal	An item to be serialized to an array of bytes to be E2E protected. For E2E protection I-Signals must be assigned to an I-Signal group.
I-Signal group	A set of I-Signals to be E2E protected. E2E protection is defined on the level of I-Signal groups.
Data element	Data exchanged between sender and receiver. The E2E transformer maps data elements 1-to-1 to I-signals or I-signal groups.
Variant	Transformer variant defined by a configuration of TransformationTechnology and EndToEndTransformationDescription. E2E transformers based on the same variant use the same E2E profile (see constr_03313 in [4]).
TransformationTechnology	TransformationTechnology is a specific class defined in [4]. This class holds general config parameters of a transformer (not only for E2E transformers).
EndToEndTransformation-Description	Specific class defined that contains about 30 parameters for E2E configuration (see [4]). It is referenced by the TransformationTechnology.
Inverse transformer	A transformer on receiver side, which reverses transformation steps applied on sender side
DEM	Diagnostic Event Manager
E2E	End-To-End Protection

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

3 Related documentation

3.1 Input documents & related standards and norms

[1] Requirements on Transformer
AUTOSAR_CP_RS_Transformer

[2] General Specification of Transformers
AUTOSAR_CP_ASWS_TransformerGeneral

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] System Template
AUTOSAR_CP_TPS_SystemTemplate

[5] E2E Protocol Specification
AUTOSAR_FO_PRS_E2EProtocol

[6] Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_CP_SWS_E2ELibrary

[7] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[8] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[9] SOME/IP Protocol Specification
AUTOSAR_FO_PRS_SOMEIPProtocol

[10] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

3.2 Related specification

AUTOSAR provides a General Specification on Transformers [2, ASWS Transformer General], which is also valid for E2E Transformer.

Thus, the specification [2, 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 [5, 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 [6, 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 [7, SWS BSW General] and [2, ASWS TransformerGeneral].

6 Requirements Tracing

Requirement	Description	Satisfied by
[RS_E2E_08528]	E2E protocol shall provide different E2E profiles	[SWS_E2EXf_00047]
[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_00047] [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_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_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_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_00206] [SWS_E2EXf_00207] [SWS_E2EXf_00208]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_E2EXf_00156]

Table 6.1: Requirements Tracing

7 Functional specification

E2E transformer is responsible for protecting safety-related data elements. On the data providing side E2E transformer E2E-protects the data. On the reception side E2E transformer E2E-checks the data, providing the result of the E2E-checks through RTE to SW-C. E2E transformer on the receipient side is called inverse E2E transformer. E2E transformer is invoked by RTE.

All algorithms are provided by [6, SWS E2ELibrary] (protect, check, forward, state machine). E2E transformer invokes E2E Library functions and provides configuration and state to be used.

It is transparent to the communicating SW-Cs whether the data on the bus is E2E protected or not, regardless of whether a receiving SW-C reads the transformer return codes or not.

Each I-signal group to be E2E-protected or E2E-tested requires at least one E2E transformer or inverse E2E transformer function.

E2E transformer is generated to a high extent, where both configuration data structures and functions are generated. The configuration input can be found in [4, System Template] and [8, Software Component Template]. No specific ECU configuration is required for the E2E transformer, since its entire configuration is based on the E2ETransformationDescription, the E2ETransformationISignalProps and the E2ETransformationComSpecProps. Thus the generic ECU configuration of the [2, ASWS Transformer General] is sufficient.

[SWS_E2EXf_00161]

Upstream requirements: RS_E2E_08538

⌈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

⌋

E2E transformer supports the following types of communication:

- *Signal based* communication
- *Service oriented* communication with events (Publisher/Subscriber)
- *Service oriented* communication with methods (client/server architecture)

Additionally a translation between signal based and service oriented communication is supported:

- Signal to service translation

Note: E2E fields are handled in the same way as events in the case of notifiers or methods in the case of get/set requests.

For *signal based* communication the sender/receiver principle is used to transmit E2E protected data. At the reception side E2E checks are applied on the received message.

For *service oriented communication* with events the E2E protection is adopted to a publisher/subscriber pattern. The E2E checks applied in general are the same as for *signal based* communication.

For *service oriented communication* with methods the E2E protection is adopted to a client/server pattern. For

1. a request (whether a response is expected or not), the E2E checks applied in general are the same as for *signal based* communication.
2. the transmission of a response to the request, an additional communication step is to be considered. Based on the check result of the request either a normal response or an error response is replied as E2E protected data.
3. a response (whether it is a normal response or an error response), the E2E checks applied in general are the same as for *signal based* communication. Additionally, a deadline monitoring could be added on the level of the receiving application.

Signal to service translation implies a change of the communication type within the network. While the provided data are signal based messages, the recipients expect service-oriented messages or vice versa. With signal to service translation a mapping between the two different communication types as well as a mapping of the E2E status is provided. For details on signal to service translation see Section 9.4.

The E2E handling is generally the same for the different types of communication but it differs in detail. These differences are shown in the flowcharts in Chapter 7.7. The following colours are used to highlight specifics for types of communication.

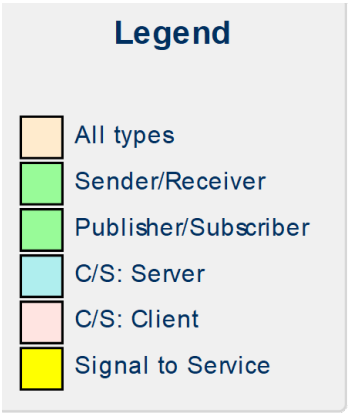

Figure 7.1: Legend to colour scheme

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>.

The pattern <transformerId> is defined in [SWS_Xfrm_00062] of [2, 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.

7.3 Configuration

A transformer configuration contains settings of all parameters used by E2E transformer functions e.g. profile names, tolerance values for E2E checks or dataID mode. For details see [4, System Template].

The following parts form three stages of transformer configuration:

- **EndToEndTransformationDescription**

  defines the E2E configuration profiles, valid for several ISignals (see [TPS_SYST_02275] in [4, System Template])

- **EndToEndTransformationISignalProps**

  defines the configuration options valid for a specific referenced ISignal (see [TPS_SYST_02275] in [4, System Template])

- **EndToEndTransformationComSpecProps**

  defines the override configuration options valid for the port to which the Receiver ComSpec belongs (see [TPS_SYST_02275] in [4, System Template]).

7.3.1 Precedence of configuration attributes

It is possible that there are several software components receiving independently data elements that are created (deserialized) from the same I-PDU. If some software components have adjusted/special configuration values, related to the tolerances of the E2E state machine, e.g. bigger tolerances, attributes of **EndToEndTransformationComSpecProps** are used.

[SWS_E2EXf_00134]

Upstream requirements: RS_E2E_08538

⌈The configuration options in EndToEndTransformationComSpecProps shall have precedence over the options in EndToEndTransformationDescription and EndToEnd TransformationISignalProps.⌋

That means:

Configuration options in EndToEndTransformationComSpecProps override the configuration options in EndToEndTransformationDescription and EndToEndTransformation ISignalProps. However the attributes of *EndToEndTransformationDescription* do not cover all attributes of *EndToEndTransformationComSpecProps* and vice versa.

7.3.2 Disable E2E check

If a software component do not need to evaluate the E2E protection, e.g. for QM software components, then the E2E check can be disabled via **EndToEndTransformationComSpecProps.disableEndToEndCheck**.

[SWS_E2EXf_00154]

Upstream requirements: RS_E2E_08538

⌈If configuration option EndToEndTransformationComSpecProps.disableEndToEnd Check 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.⌋

7.3.3 Configuration variants

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

[SWS_E2EXf_00090]

Upstream requirements: RS_E2E_08538

⌈In case of post-build-selectable configuration, the variants shall be named according to the configuration attribute PredefinedVariant.shortName. This means:

<v> = PredefinedVariant.shortName.⌋

[SWS_E2EXf_00089]

Upstream requirements: RS_E2E_08538

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

<v>= emtpy (NULL string).⌋

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>).

7.4 Generated structure types

Based on the E2E transformer configuration (described in [4, System Template], [8, Software Component Template] and the generated ECU configuration (described in [2, General Specification of Transformers]), the corresponding C structures are generated as described below.

7.4.1 Overall configuration and state of E2E transformer

[SWS_E2EXf_00011]

Upstream requirements: RS_E2E_08538

⌈The E2E transformer shall generate the following data structure, to store the configuration of E2E transformer module:

E2EXf_ConfigStruct_<v>(of type E2EXf_ConfigType)⌋

[SWS_E2EXf_00125]

Upstream requirements: RS_E2E_08538

⌈The E2E transformer shall derive the required number of independent state data resources of types E2E_PXXProtectStateType, E2E_PXXCheckStateType, and E2E_SMCheckStateType to perform E2E Protection within the E2E transformer module from the number of E2E-protected data uniquely identified with <transformerId>, protected by profile PXX.⌋

7.4.2 Configuration and state of each E2E-protected data

[SWS_E2EXf_00126]

Upstream requirements: RS_E2E_08538

⌈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).

⌋

7.5 Static initialization

7.5.1 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]

Upstream requirements: RS_E2E_08538

⌈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.

⌋

[SWS_E2EXf_00118]

Upstream requirements: RS_E2E_08538

⌈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.

⌋

[SWS_E2EXf_00087]

Upstream requirements: RS_E2E_08538

⌈The generated configuration object of type E2E_P04ConfigType, E2E_P06ConfigType, E2E_P07ConfigType, E2E_P08ConfigType, E2E_P44ConfigType, E2E_P04mConfigType, E2E_P07mConfigType, E2E_P08mConfigType, E2E_P44mConfigType 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

⌋

[SWS_E2EXf_00119]

Upstream requirements: RS_E2E_08538

⌈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

⌋

[SWS_E2EXf_00162]

Upstream requirements: RS_E2E_08538

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

- DataID = EndToEndTransformationISignalProps.dataID
- DataLength = EndToEndTransformationISignalProps.dataLength

- CounterOffset = EndToEndTransformationDescription.counterOffset
- CRCOffset = EndToEndTransformationDescription.crcOffset
- DataIDNibbleOffset = EndToEndTransformationDescription.dataIdNibbleOffset
- DataIDMode shall be set to
  - E2E_P11_DATAID_BOTH if EndToEndTransformationDescription.dataIDMode == all16Bit
  - E2E_P11_DATAID_NIBBLE if EndToEndTransformationDescription.dataIDMode == nibble
- MaxDeltaCounter = EndToEndTransformationComSpecProps.maxDeltaCounter or EndToEndTransformationDescription.maxDeltaCounter

⌋

[SWS_E2EXf_00164]

Upstream requirements: RS_E2E_08538

⌈The generated configuration object of type E2E_P22ConfigType shall be initialized according to the following:

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

⌋

[SWS_E2EXf_00088]

Upstream requirements: RS_E2E_08538

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

- WindowSizeValid = EndToEndTransformationComSpecProps.windowSizeValid or EndToEndTransformationDescription.windowSizeValid
- WindowSizeInit = EndToEndTransformationComSpecProps.windowSizeInit or EndToEndTransformationDescription.windowSizeInit
- WindowSizeInvalid = EndToEndTransformationComSpecProps.windowSizeInvalid or EndToEndTransformationDescription.windowSizeInvalid
- ClearToInvalid = EndToEndTransformationComSpecProps.clearFromValidToInvalid or EndToEndTransformationDescription.clearFromValidToInvalid

- transitToInvalidExtended = EndToEndTransformationComSpecProps.E2EProfileCompatibilityProps.transitToInvalidExtended or EndToEndTransformationDescription.E2EProfileCompatibilityProps.transitToInvalidExtended
- MinOkStateInit = EndToEndTransformationComSpecProps.minOkStateInit or EndToEndTransformationDescription.minOkStateInit
- MaxErrorStateInit = EndToEndTransformationComSpecProps.maxErrorStateInit or EndToEndTransformationDescription.maxErrorStateInit
- MinOkStateValid = EndToEndTransformationComSpecProps.minOkStateValid or EndToEndTransformationDescription.minOkStateValid
- MaxErrorStateValid = EndToEndTransformationComSpecProps.maxErrorStateValid or EndToEndTransformationDescription.maxErrorStateValid
- MinOkStateInvalid = EndToEndTransformationComSpecProps.minOkStateInvalid or EndToEndTransformationDescription.minOkStateInvalid
- MaxErrorStateInvalid = EndToEndTransformationComSpecProps.maxErrorStateInvalid or EndToEndTransformationDescription.maxErrorStateInvalid

⌋

For priority of values see Section 7.3.1 "Precedence of configuration attributes".

[SWS_E2EXf_00096]

Upstream requirements: RS_E2E_08538

⌈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.⌋

7.5.2 Static Initialization of state

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

[SWS_E2EXf_00023]

Upstream requirements: RS_E2E_08538

⌈In all E2E transformer variants, the generated state objects may be left uninitialized (i.e. without providing explicit initialization values).⌋

7.6 Runtime initialization by E2EXf_Init() function

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

[SWS_E2EXf_00024]

Upstream requirements: RS_E2E_08538

⌈In post-build-selectable variant, E2EXf_Init() shall check that Config pointer (received as function parameter) points to one of the configuration variants E2EXf_Config Struct_<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.⌋

7.6.2 Runtime initialization of State

[SWS_E2EXf_00021]

Upstream requirements: RS_E2E_08538

⌈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.

⌋

[SWS_E2EXf_00158]

Upstream requirements: RS_E2E_08538

⌈The E2EXf_Init() function shall initialize the internal state data resources of E2E functions forward, protect check and state machine of E2E transformer.⌋

[SWS_E2EXf_00159]

Upstream requirements: RS_E2E_08538

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

[SWS_E2EXf_00130]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00132]

Upstream requirements: RS_E2E_08538

⌈In case of deinitialization (invocation of E2EXf_DeInit()), the module initialization state shall be set to FALSE.⌋

7.7 Normal operation

[SWS_E2EXf_00133]

Upstream requirements: RS_E2E_08538

⌈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.⌋

7.7.1 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 processing 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 processing means that separate buffer, one input buffer and a separate output buffer, are used.

7.7.2 Transformer and E2E protection

E2E transformer invokes E2E Library protect() function via API E2EXf_<transformerId>(protect-function). This way configuration data and data to be E2E protected are provided to E2E Library functions.

Depending on the type of communication, the flow of E2EXf_<transformerId>() varies.

A general overview of this function is provided in 7.7.2.1. Details on specific parts are provided in 7.7.2.2 to 7.7.2.10. Communication type specific parts are coloured as described in Figure 7.1.

7.7.2.1 E2EXf_<transformerId>(protect-function)

[SWS_E2EXf_00020]

Upstream requirements: RS_E2E_08538

⌈The function E2EXf_<transformerId> shall be generated for each sent E2E-protected data element and each protected C/S operation (<transformerId>).⌋

The following figure provides an activity diagram of the functionality provided by the API function E2EXf_<transformerId>.

Details on

- E2EXf_input_checks
- E2EXf_handling_P01_P02
- E2EXf handling buffer and header
- E2EXf handling communication type
- E2EXf_store_request_counter
- E2EXf_handling_PXXm_client
- E2EXf_handling_PXXm_server
- E2EXf_set_response_counter
- E2EXf data transformation

are provided in the following subchapters.

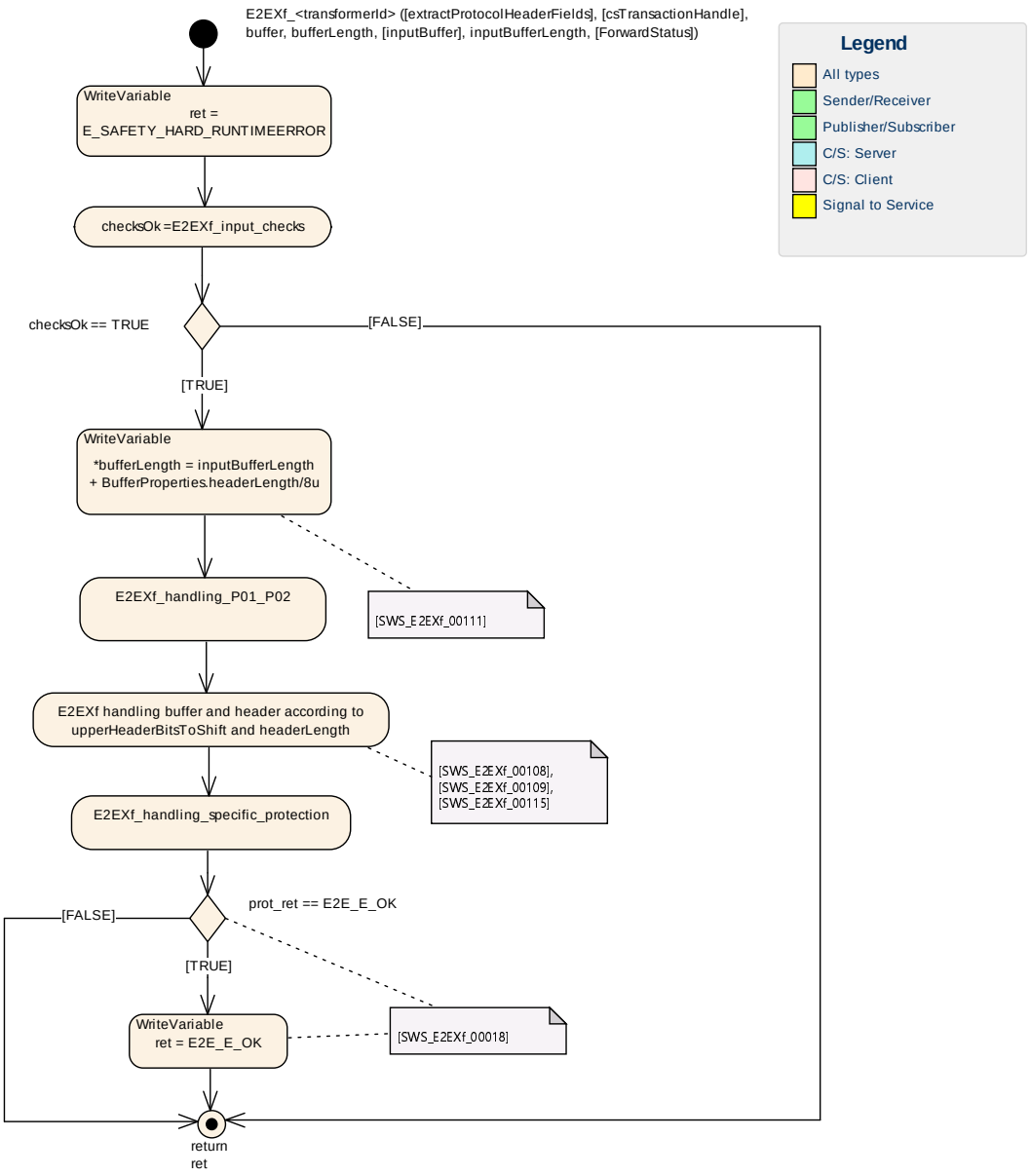

Figure 7.2: E2EXf_<transformerId> function overview

7.7.2.2 E2EXf_input_checks

E2EXf_input_checks performs some checks on input variables (e.g. buffer sizes, pointers) and returns the result as shown in the following figure.

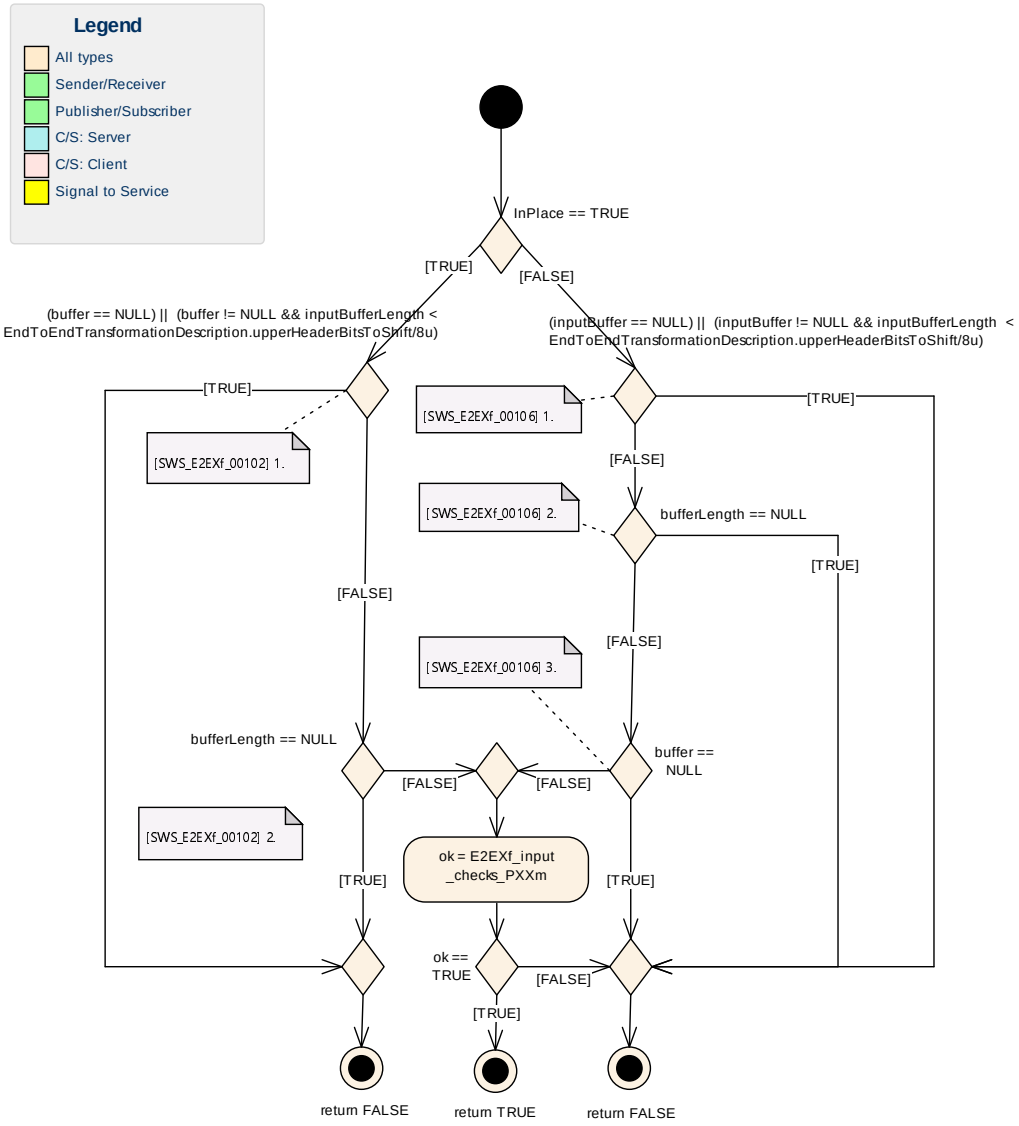

Figure 7.3: E2EXf_input_checks()

[SWS_E2EXf_00102]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00106]

Upstream requirements: RS_E2E_08538

⌈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.⌋

For method specific E2E profiles additional checks as shown in the following figure are to be applied.

[SWS_E2EXf_00180]

Upstream requirements: RS_E2E_08538

⌈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.⌋

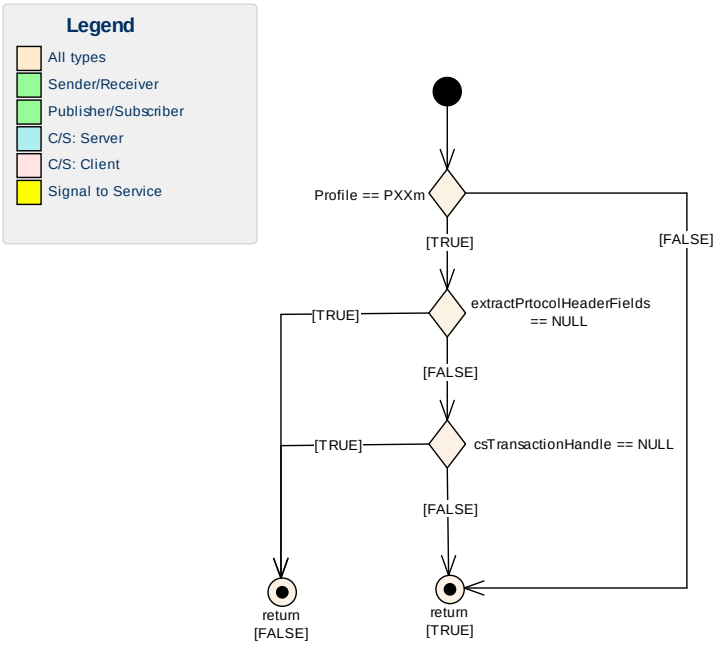

Figure 7.4: E2EXf_input_checks_PXXm()

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.

7.7.2.3 E2EXf_handling_P01_P02

E2E transformer implements a specific handling of legacy profiles P01 and P02, as Library functions of these profiles were developed before the transformer approach was introduced into AUTOSAR and therefore provide different APIs. The handling of P01 and P02 is shown in the following figure.

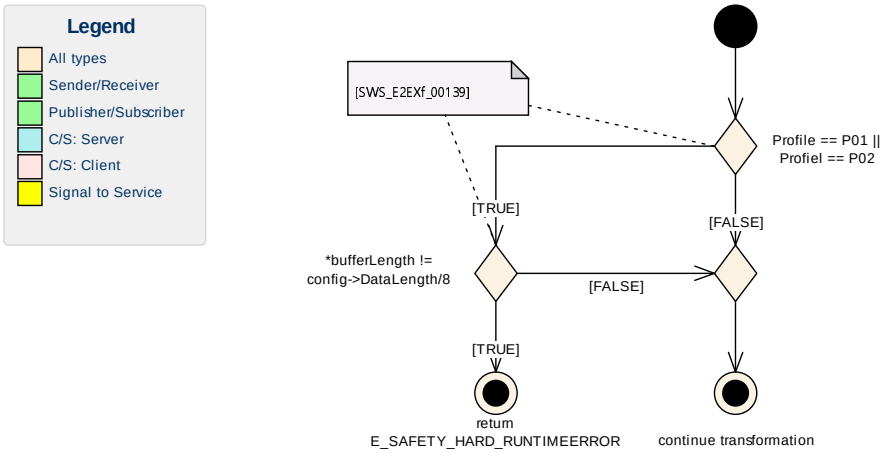

Figure 7.5: E2EXf_handling_P01_P02

[SWS_E2EXf_00139]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00155]

Status: DRAFT
Upstream requirements: RS_E2E_08538

⌈If (((PXX = 01 &&dataIDMode != nibble) || (PXX == 02) || (PXX = 11 &&dataIDMode != nibble) || (PXX == 22)) && BufferProperties.headerLength == 16 [bits]), the function E2EXf_<transformerId>() shall, before calling E2E_PXXProtect() or E2E_PXXForward(), set 0xF in buffer at the bit offset (EndToEndTransformationDescription.crcOffset+12 for profiles P01 and P11 and EndToEndTransformationDescription.offset+12 for profiles P02 and P22).⌋

7.7.2.4 E2EXf Handling buffer and header

E2E transformer can be configured for the usage of in-place or out-of-place buffer. Both kinds of buffer handling and their implication for header handling are described in the following.

[SWS_E2EXf_00108]

Upstream requirements: RS_E2E_08538

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

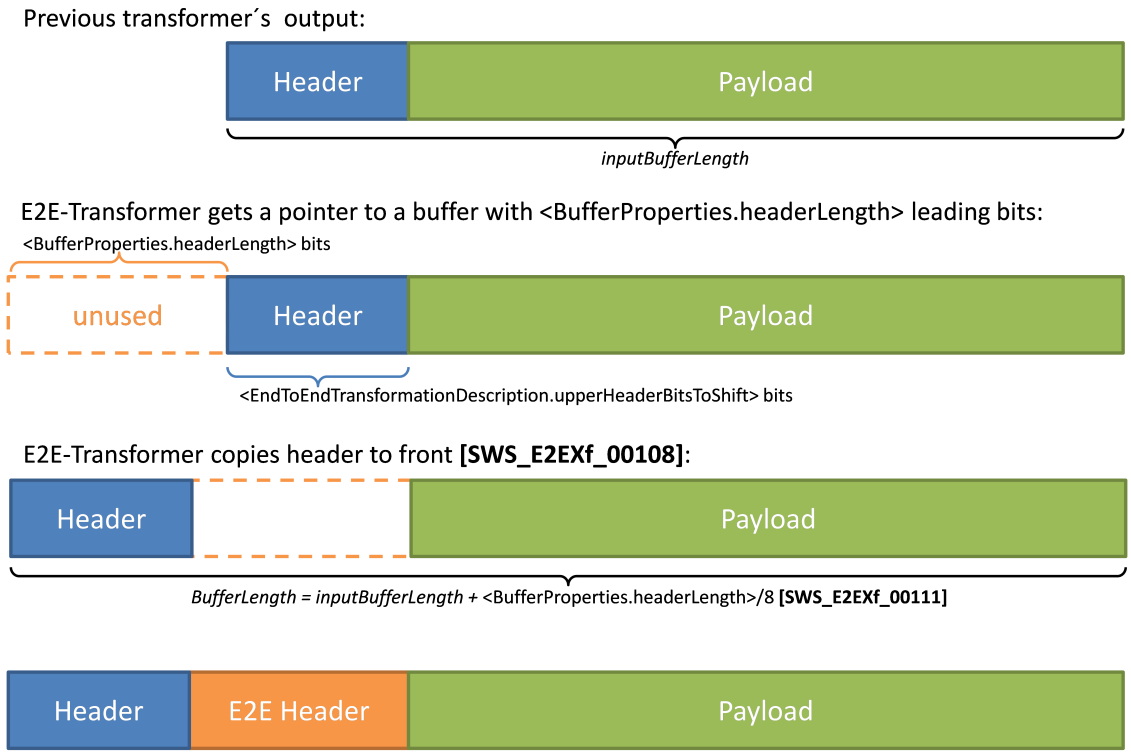

Figure 7.6: Buffer Handling of E2E_Transformer_<transformerID>

Figure 7.6 illustrates the buffer handling done by API function E2EXf_<transformer Id>for In-Place.

[SWS_E2EXf_00109]

Upstream requirements: RS_E2E_08538

⌈If (EndToEndTransformationDescription.upperHeaderBitsToShift > 0), out-of-place E2EXf_<transformerId> shall copy the first upper HeaderBitsToShiftbits from input Buffer 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).⌋

[SWS_E2EXf_00115]

Upstream requirements: RS_E2E_08538

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

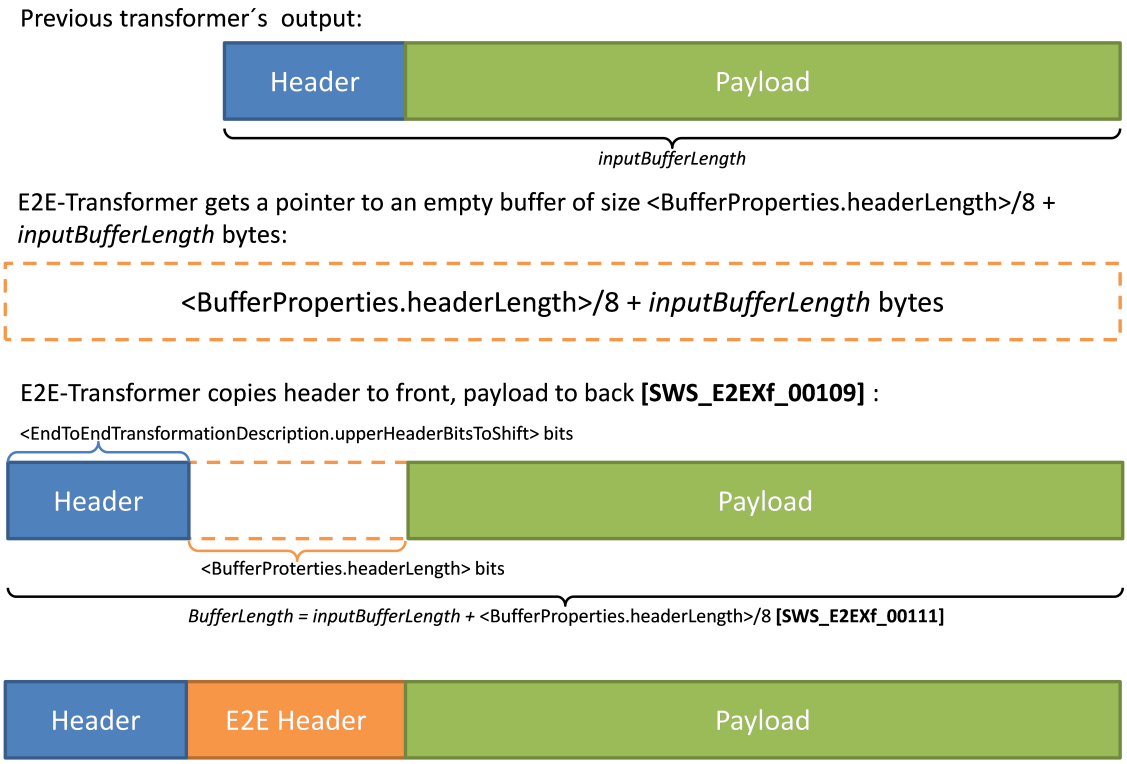

Figure 7.7: Header Shift Out of Place

Figure 7.7 illustrates the buffer handling done by API function E2EXf_<transformer Id>for Out-of-place.

[SWS_E2EXf_00111]

Upstream requirements: RS_E2E_08538

⌈E2EXf_<transformerId> shall set *bufferLength = inputBufferLength + BufferProperties.headerLength/8.⌋

7.7.2.5 E2EXf_handling_specific_protection

The specific protection describes different activities depending on different communication types (see introduction of Chapter 7).

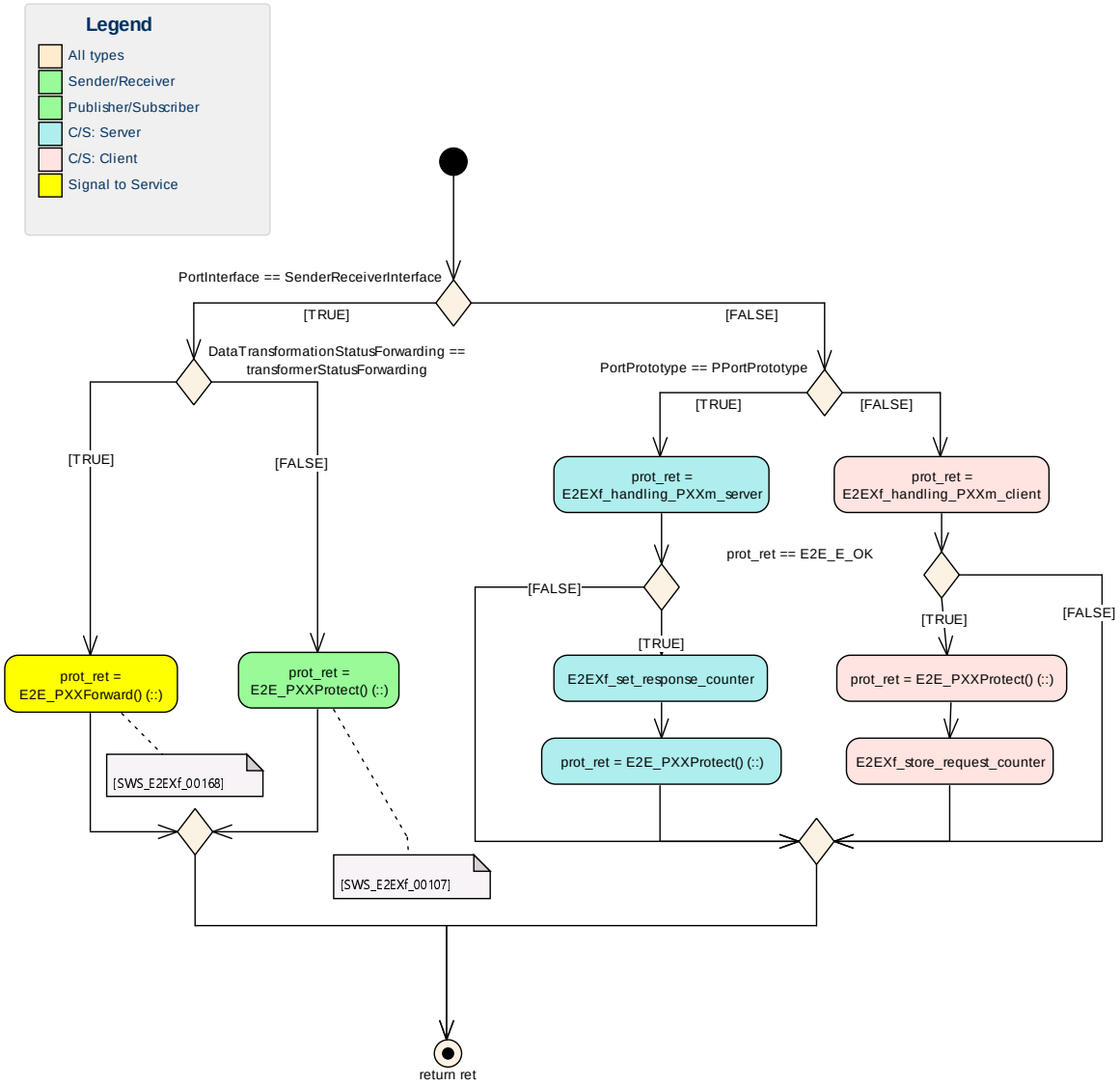

Figure 7.8: E2EXf_Communication_Type_Specific_Protection

7.7.2.6 E2EXf_store_request_counter

The message counter of a response message shall be the same as the counter of the corresponding request message. Therefore, the requestCounter is stored and used as an input to the check function. The request counter is stored profile depended as shown in the following figure.

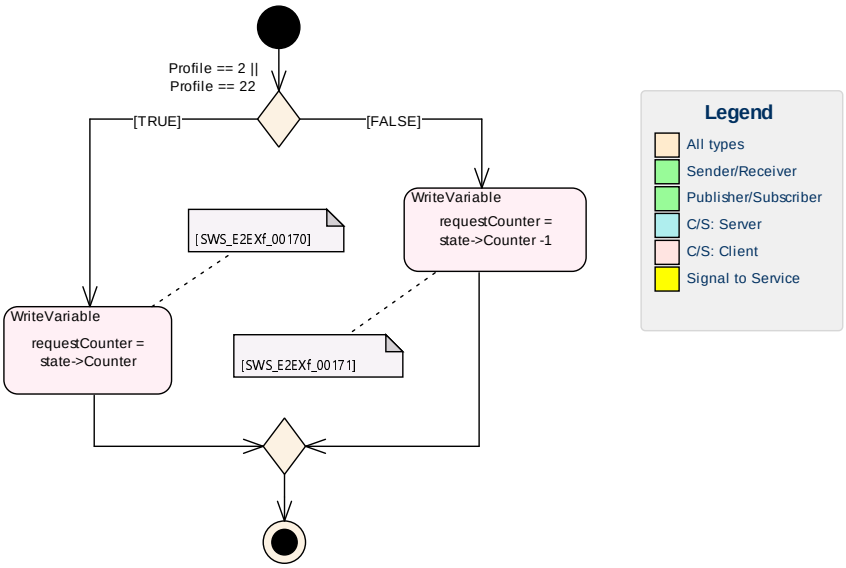

Figure 7.9: E2EXf_store_request_counter

[SWS_E2EXf_00170]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00171]

Upstream requirements: RS_E2E_08538

⌈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.⌋

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.

7.7.2.7 E2EXf_handling_PXXm (client)

E2E transformer implements a specific handling of method profiles PXXm, as library functions require additional parameters and therefore provide different APIs. The protection of PXXm profiles on client side is shown in the following figure.

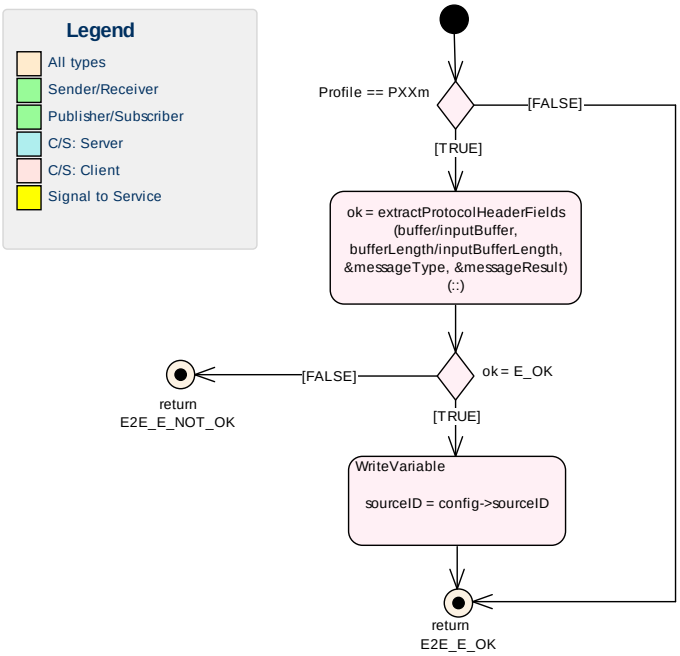

Figure 7.10: E2EXf_handling_PXXm - client

[SWS_E2EXf_00181]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00182]

Upstream requirements: RS_E2E_08538

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

[SWS_E2EXf_00183]

Upstream requirements: RS_E2E_08538

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

[SWS_E2EXf_00184]

Upstream requirements: RS_E2E_08538

⌈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).⌋

7.7.2.8 E2EXf_handling_PXXm (server)

E2E transformer implements a specific handling of method profiles PXXm, as Library functions require additional parameters and therefore provide different APIs. The protection of PXXm profiles on server side is shown in the following figure.

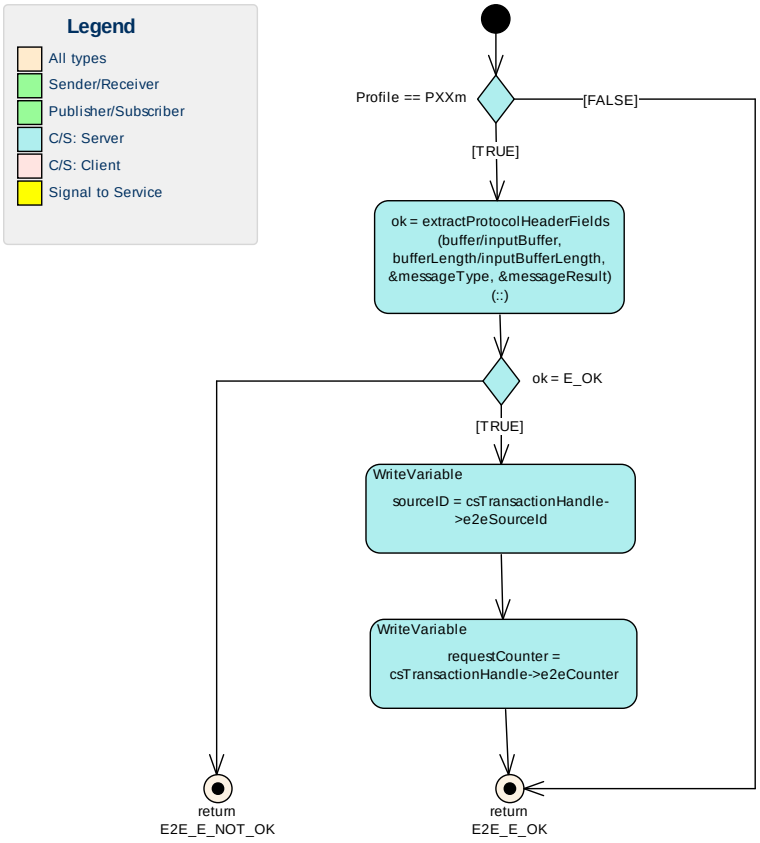

Figure 7.11: E2EXf_handling_PXXm - server

[SWS_E2EXf_00185]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00186]

Upstream requirements: RS_E2E_08538

⌈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 messageType variable, and the address of a local messageResult variable as parameters.⌋

[SWS_E2EXf_00187]

Upstream requirements: RS_E2E_08538

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

[SWS_E2EXf_00188]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00189]

Upstream requirements: RS_E2E_08538

⌈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 request Counter to the value of the e2eCounter element of the csTransactionHandle.⌋

7.7.2.9 E2EXf_set_response_counter

The message counter of a response message shall be the same as the counter of the corresponding request message. Therefore, the stored receivedRequestCounter is used as an input to the protect function. The response counter is set profile depended as shown in the following figure. In case of profile 02 or 22 the receivedRequestCounter needs to be decremented since protect() and forward() functions of these profiles increment the counter prior to adding it to the message.

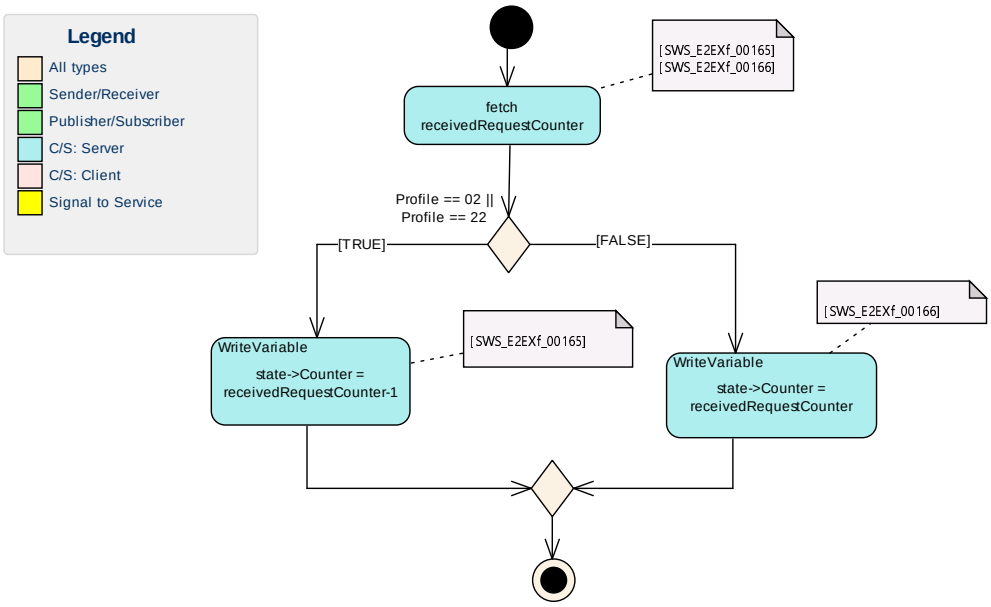

Figure 7.12: E2EXf_set_response_counter

[SWS_E2EXf_00165]

Upstream requirements: RS_E2E_08538

⌈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.⌋

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]

Upstream requirements: RS_E2E_08538

⌈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.⌋

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.

7.7.2.10 E2EXf Data Transformation

E2E transformer implements a specific handling for signal to service translation or vise versa. The forwarding() function replaces the protect() function to forward the result of an E2E check() in a new E2E protected message.

[SWS_E2EXf_00107]
Status: DRAFT
Upstream requirements: RS_E2E_08538

⌈If DataTransformationStatusForwarding is set to noTransformerStatusForwarding 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 buffer Length (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.⌋

[SWS_E2EXf_00190]
Upstream requirements: RS_E2E_08538

⌈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.⌋

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]

Status: DRAFT
Upstream requirements: RS_E2E_08538

⌈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 buffer Length (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 [SWS_E2EXf_00208]).

⌋

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

[SWS_E2EXf_00191]

Upstream requirements: RS_E2E_08538

⌈If DataTransformationStatusForwarding is set to transformerStatusForwarding 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.⌋

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]

Status: DRAFT
Upstream requirements: RS_E2E_08538

⌈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.⌋

7.7.2.11 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]

Upstream requirements: 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

Check Status Mapping of forwardedCode

⌋

7.7.3 Transformer and E2E check

E2E transformer invokes E2E Library check() function via API E2EXf_<transformerId>(check-function). This way configuration data and data to be E2E protected are provided to E2E Library function's. Depending on the type of communication present, the flow of E2EXf_<transformerId>() varies. A general overview of this function is provided in 7.7.3.1. Details on specific parts are provided in 7.7.3.2 to 7.7.3.9 Communication type specifc parts are coloured as described in Figure 7.1.

7.7.3.1 E2EXf_Inv_<transformerId>(check-function)

[SWS_E2EXf_00025]

Upstream requirements: RS_E2E_08538

⌈The function E2EXf_Inv_<transformerId> shall be generated for each received E2E-protected data element and each protected C/S operation (<transformerId>).⌋

The following figure provides an activity diagram of the functionality provided by the API function E2EXf_Inv_<transformerId>.

Details on

- E2EXf_Inv_input_checks
- E2EXf_Inv_handling_P01_P02
- E2EXf_Inv_handling_P01_P02_forceConstantMaxDeltaCounter
- E2EXf_Inv_handling communication type
- E2EXf_Inv_handling_PXXm (client)
- E2EXf_Inv_handling_PXXm (server)
- E2EXf_Inv_set_check_counter
- E2EXf_Inv_store_request_counter
- E2EXf_Inv_handle_Statemachine

are provided in the following subchapters.

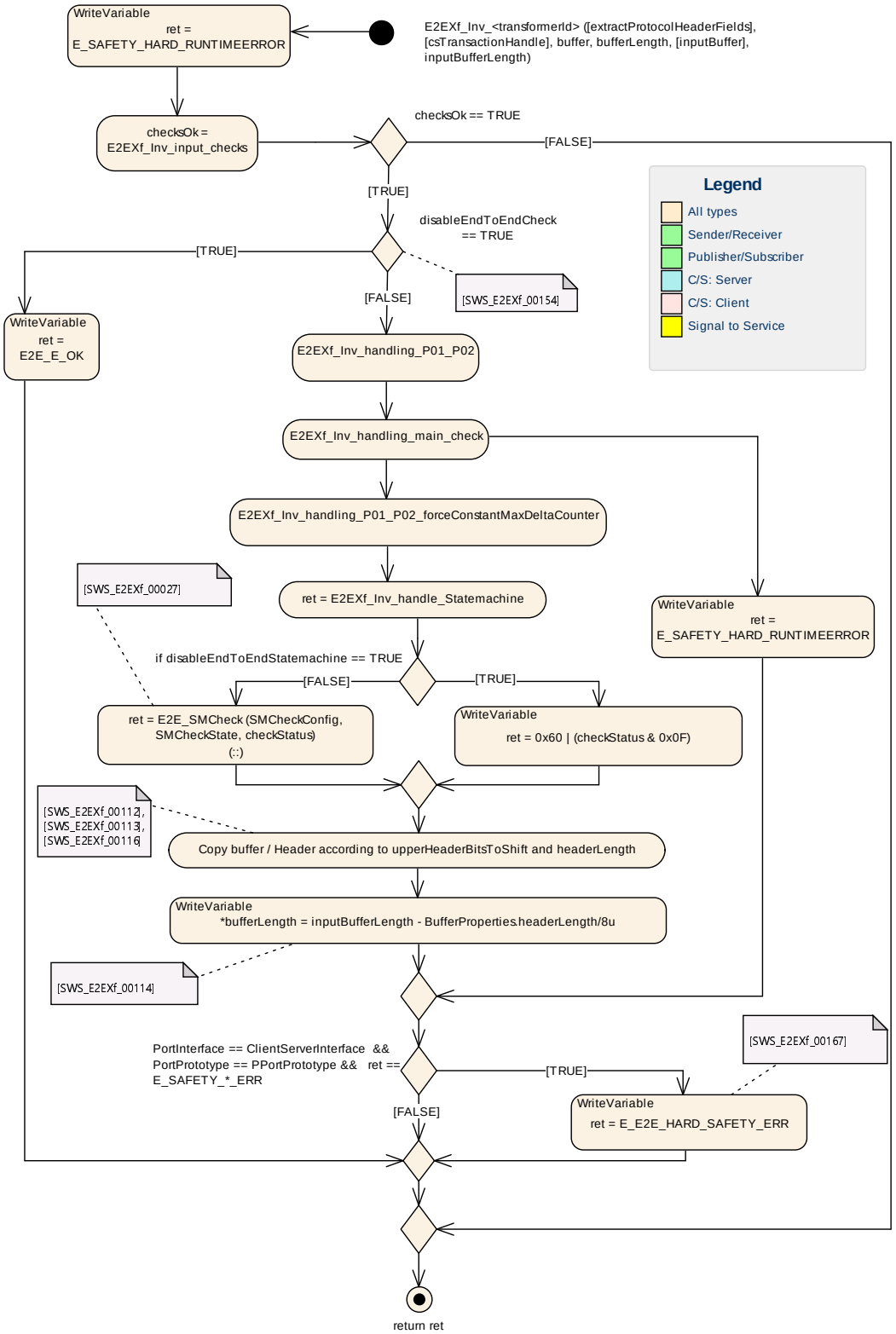

Figure 7.13: E2EXf_Inv_<transformerId>function overview

7.7.3.2 E2EXf_Inv_input_checks

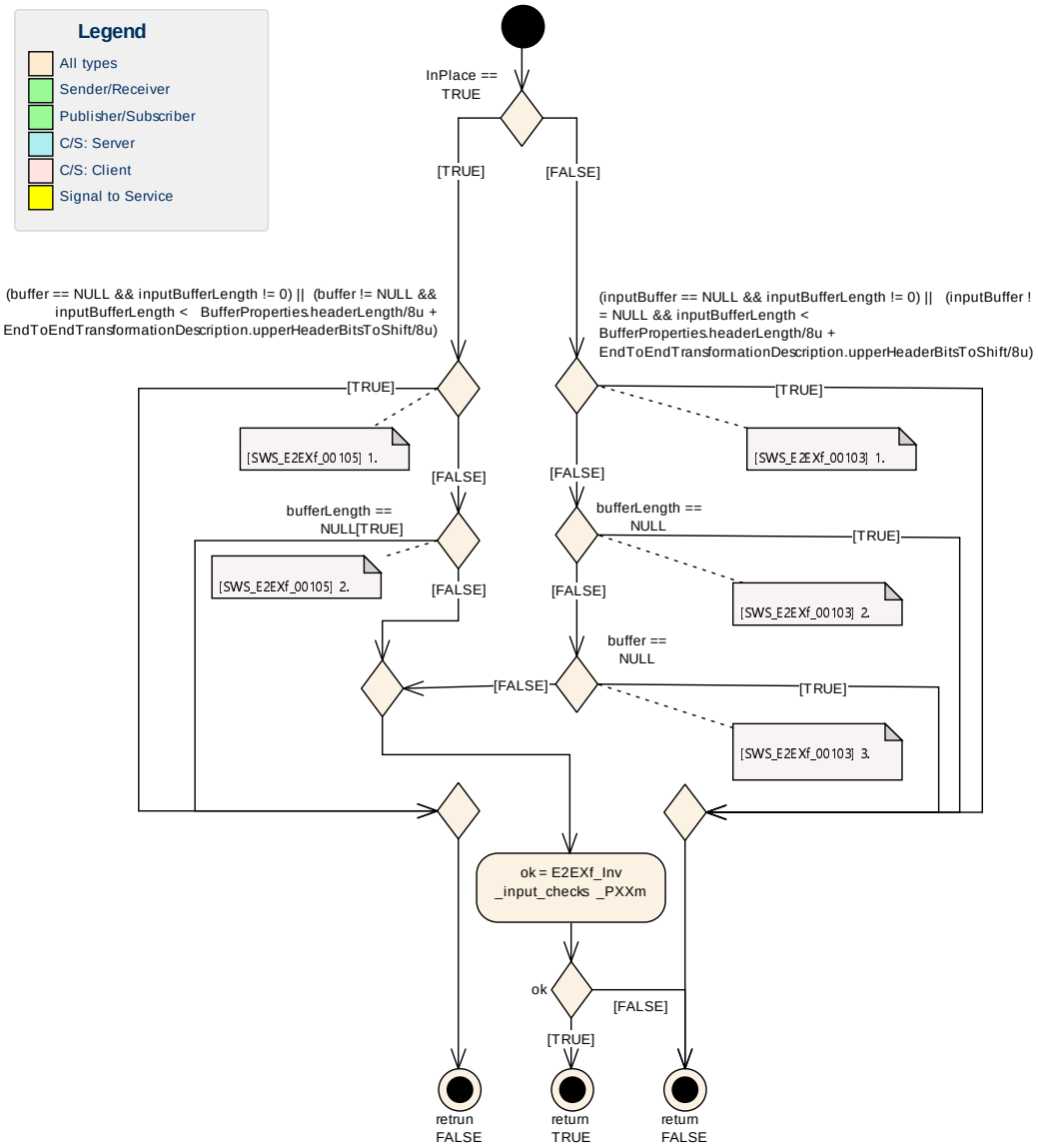

Figure 7.14: E2EXf_Inv_input_checks

[SWS_E2EXf_00105]

Upstream requirements: RS_E2E_08538

⌈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 + End ToEndTransformationDescription.upperHeaderBitsToShift/8u)

2. bufferLength == NULL.

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

[SWS_E2EXf_00103]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00192]

Upstream requirements: RS_E2E_08538

⌈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.⌋

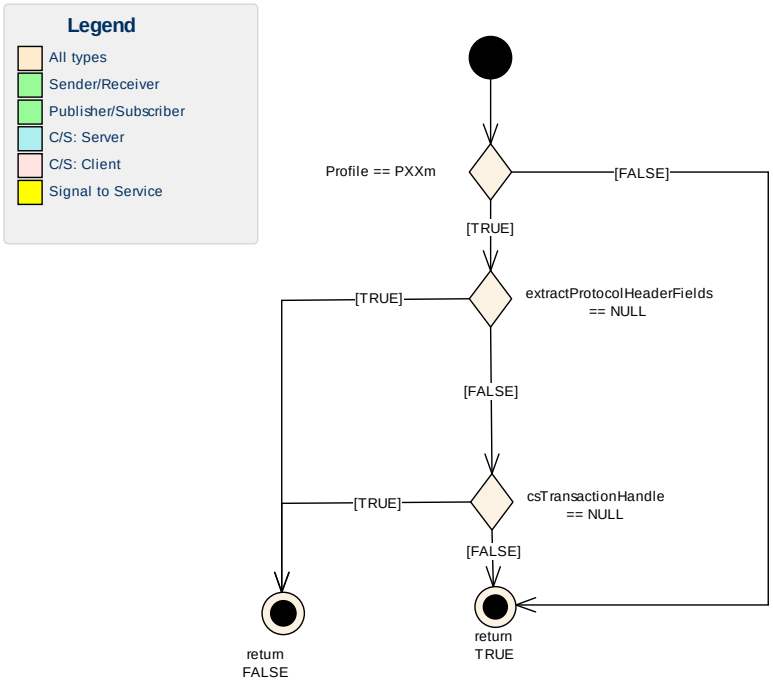

Figure 7.15: E2EXf_Inv_input_checks_PXXm

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.

Table 7.1: Return values of input checks

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]

Upstream requirements: RS_E2E_08538

⌈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.

⌋

[SWS_E2EXf_00123]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00141]

Upstream requirements: RS_E2E_08538

⌈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.

⌋

7.7.3.3 E2EXf_Inv_handling_P01_P02

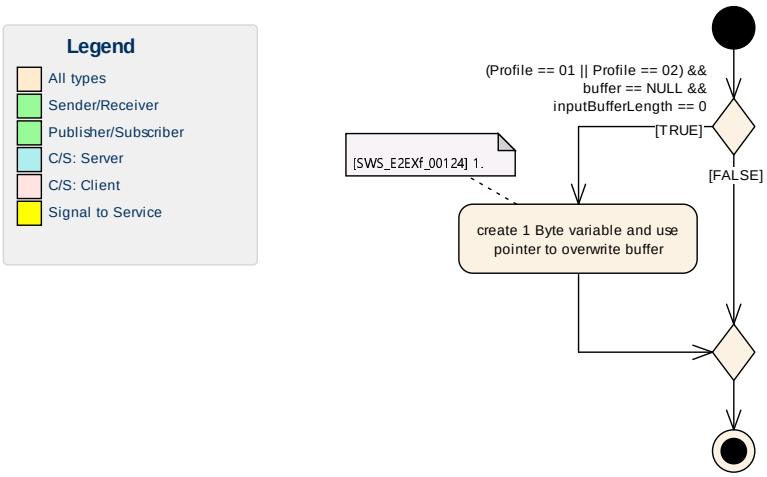

Figure 7.16: E2EXf_Inv_handling_P01_P02

[SWS_E2EXf_00124]

Upstream requirements: RS_E2E_08538

⌈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.⌋

7.7.3.4 E2EXf_Inv_handling_P01_P02_forceConstantMaxDeltaCounter

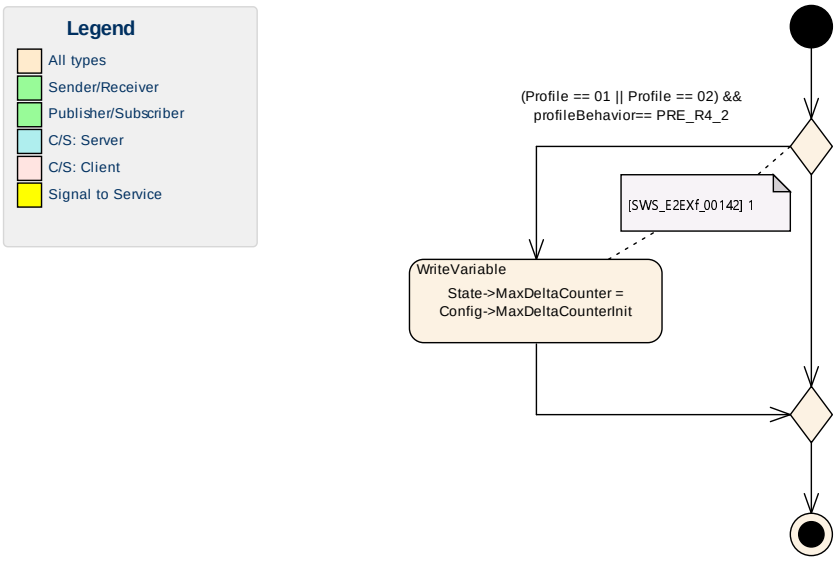

Figure 7.17: E2EXf_Inv_handling_P01_P02_forceConstantMxDeltaCounter

[SWS_E2EXf_00142]

Upstream requirements: RS_E2E_08538

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

[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]

Upstream requirements: RS_E2E_08538

⌈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).⌋

7.7.3.5 E2EXf_Inv_handling_main_check

The specific check describes different activities depending on different communication types (see introduction of Chapter 7).

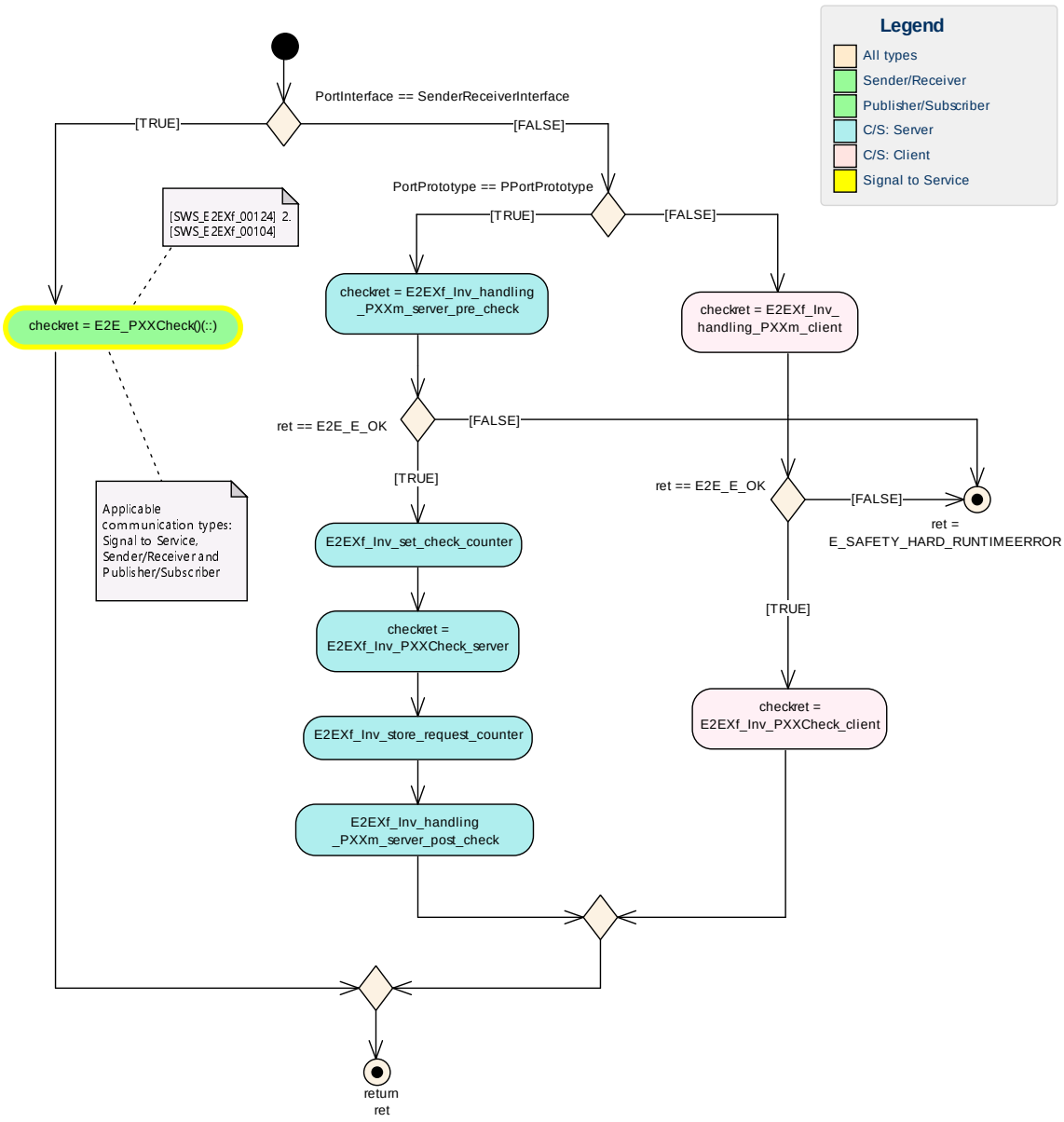

Figure 7.18: E2EXf_Inv_handling_main_check

7.7.3.6 E2EXf_Inv_handling_PXXm

[SWS_E2EXf_00193]

Upstream requirements: RS_E2E_08538

⌈For profiles P04m, P07m, P08m and P44m the in-place E2EXf_Inv_<transformer Id> 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.⌋

[SWS_E2EXf_00194]

Upstream requirements: RS_E2E_08538

⌈For profiles P04m, P07m, P08m and P44m the out-of-place E2EXf_Inv_<transformer Id> on the client-side shall call the extractProtocolHeaderFields() function passing the inputBuffer, the inputBufferLength, the address of a local variable named message Type, and the address of a local variable named messageResult as parameters.⌋

[SWS_E2EXf_00196]

Upstream requirements: RS_E2E_08538

⌈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).⌋

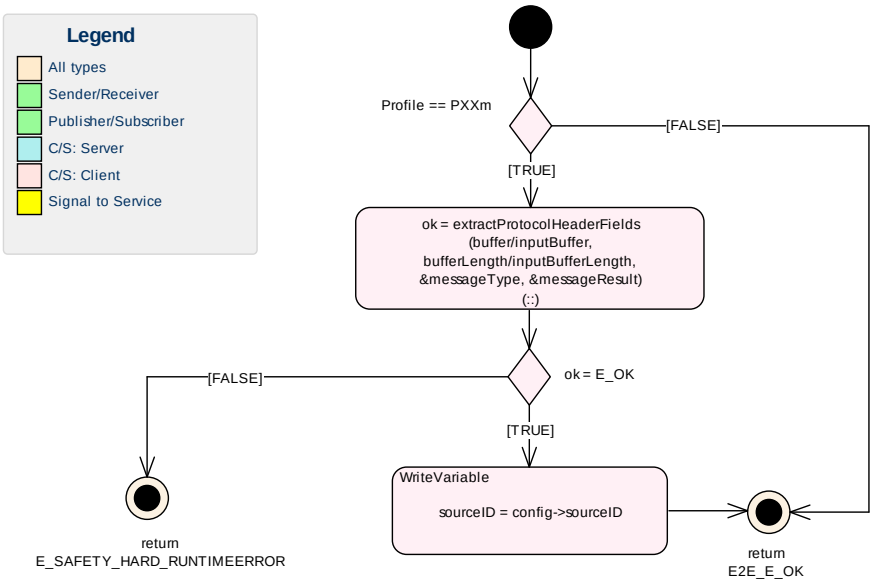

Figure 7.19: E2EXf_Inv_handling_PXXm - client

[SWS_E2EXf_00197]

Upstream requirements: RS_E2E_08538

⌈For profiles P04m, P07m, P08m and P44m the in-place E2EXf_Inv_<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.⌋

[SWS_E2EXf_00198]

Upstream requirements: RS_E2E_08538

⌈For profiles P04m, P07m, P08m and P44m the out-of-place E2EXf_Inv_<transformer Id> on the server-side shall call the extractProtocolHeaderFields() function passing the

inputBuffer, the inputBufferLength, the address of local messageType variable, and the address of a local messageResult variable as parameters.⌋

[SWS_E2EXf_00199]

Upstream requirements: RS_E2E_08538

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

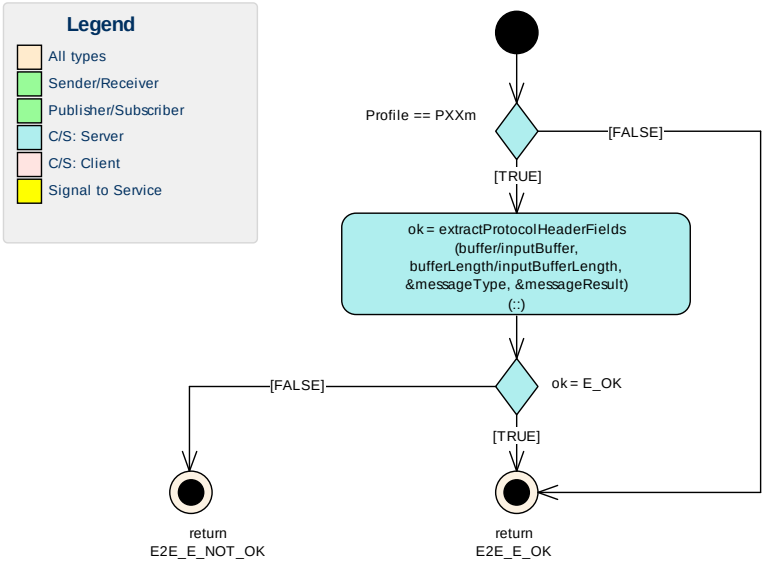

Figure 7.20: E2EXf_Inv_handling_PXXm - server (pre_check)

[SWS_E2EXf_00200]

Upstream requirements: RS_E2E_08538

⌈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).

⌋

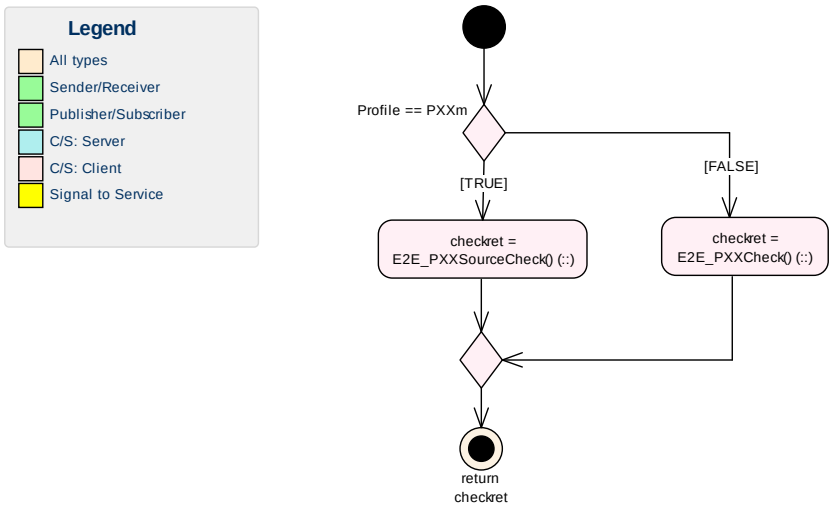

Figure 7.21: E2EXf_Inv_handling_PXXCheck - client(check)

[SWS_E2EXf_00201]

Upstream requirements: RS_E2E_08538

⌈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).

⌋

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().

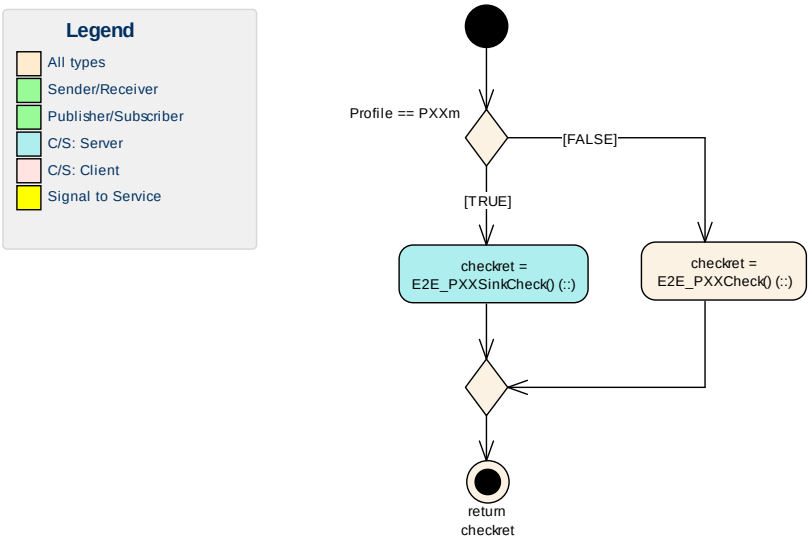

Figure 7.22: E2EXf_Inv_handling_PXXCheck - server

[SWS_E2EXf_00202]

Upstream requirements: RS_E2E_08538

⌈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 e2eSourceId element of the csTransactionHandle to the value of the local variable sourceID.⌋

[SWS_E2EXf_00203]

Upstream requirements: RS_E2E_08538

⌈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 e2eCounter element of the csTransactionHandle to the value of the local variable receivedRequestCounter.⌋

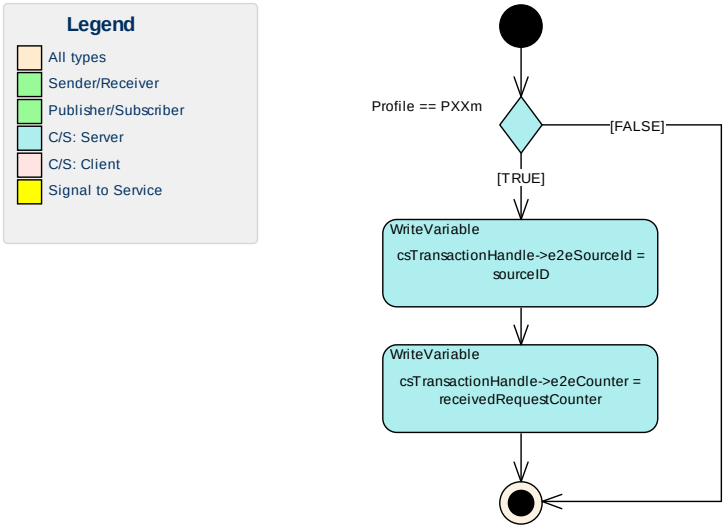

Figure 7.23: E2EXf_Inv_handling_PXXm - server (post_check)

7.7.3.7 E2EXf_Inv_set_check_counter

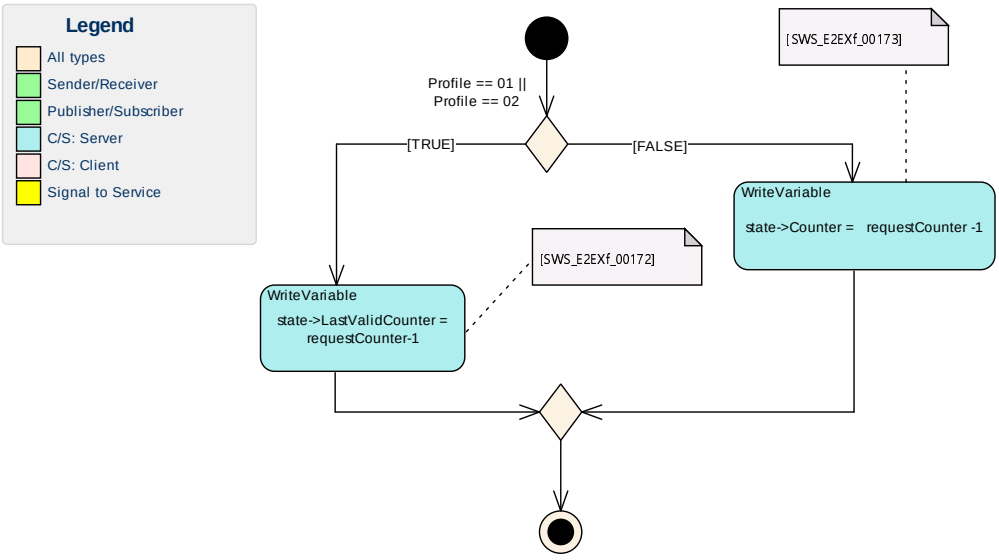

Figure 7.24: E2EXf_Inv_set_check_counter

[SWS_E2EXf_00172]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00173]

Upstream requirements: RS_E2E_08538

⌈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)

7.7.3.8 E2EXf_Inv_store_request_counter

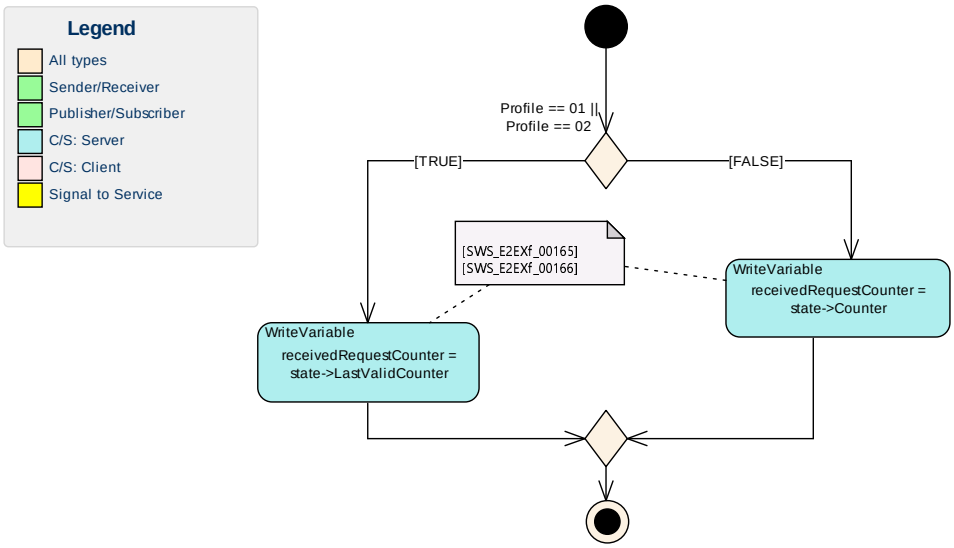

Figure 7.25: E2EXf_Inv_store_request_counter

[SWS_E2EXf_00206]

Upstream requirements: RS_E2E_08538

⌈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>.⌋

[SWS_E2EXf_00207]

Upstream requirements: RS_E2E_08538

⌈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 receivedRequest-Counter shall be set to state->Counter of E2EXf_Inv_<transformerId>.⌋

This is the counter part for requirements SWS_E2EXf_00165 and SWS_ E2EXf_00166.

7.7.3.9 E2EXf_Inv_handle_Statemachine

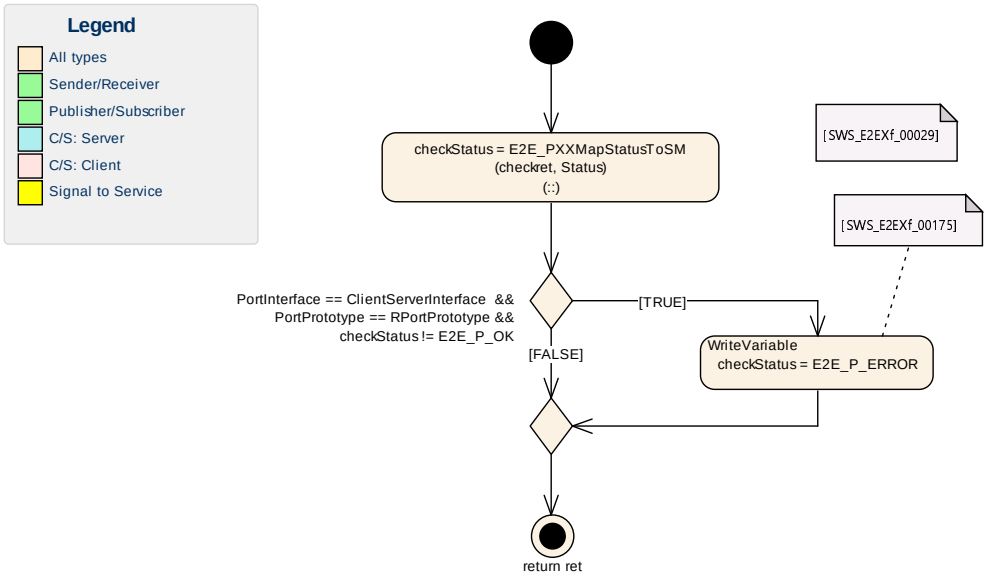

Figure 7.26: E2EXf_Inv_handle_Statemachine

[SWS_E2EXf_00029]

Upstream requirements: RS_E2E_08538

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

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

⌋

[SWS_E2EXf_00028]

Upstream requirements: RS_E2E_08538

⌈The function E2EXf_Inv_<transformerId> shall invoke the E2E_SMCheck() function if disableEndToEndStateMachine equals FALSE. It shall pass to E2E_SMCheck() 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].⌋

[SWS_E2EXf_00169]

Upstream requirements: RS_E2E_08538

⌈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.

⌋

[SWS_E2EXf_00027]

Upstream requirements: RS_E2E_08538

⌈If E2E_SMCheck() returns E2E_E_OK and disableEndToEndStateMachine 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.⌋

[SWS_E2EXf_00112]

Upstream requirements: RS_E2E_08538

⌈If (buffer != NULL &&EndToEndTransformationDescription.upperHeaderBitsToShift> 0), in-place E2EXf_Inv_<transformerId> shall copy the first upper HeaderBitsToShift-bits, in parameter buffer, in direction right by "distance" of BufferProperties.header Length.⌋

[SWS_E2EXf_00113]

Upstream requirements: RS_E2E_08538

⌈If (inputBuffer != NULL &&EndToEndTransformationDescription.upperHeaderBitsTo Shift> 0), out-of-place E2EXf_Inv_<transformerId> shall copy the first upper Header BitsToShiftbits from inputBuffer to buffer, and then copy the remaining part of input Buffer skipping E2E header (i.e. starting with offset upperHeaderBitsToShift+Buffer Properties.headerLength) to parameter buffer starting with the destination offset of (upperHeaderBitsToShift).⌋

[SWS_E2EXf_00116]

Upstream requirements: RS_E2E_08538

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

[SWS_E2EXf_00114]

Upstream requirements: RS_E2E_08538

⌈If inputBufferLength == 0, then E2EXf_Inv_<transformerId> shall set *bufferLength = 0, otherwise it shall set *bufferLength = inputBufferLength - BufferProperties.headerLength/8.⌋

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

[SWS_E2EXf_00167]

Upstream requirements: RS_E2E_08538

⌈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 .⌋

The handling of E_SAFETY_*_ERR errors as hard errors creates a TransformerHardErrorEvent. 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 [9, SOME/IP Protocol Specification].

[SWS_E2EXf_00175]

Upstream requirements: RS_E2E_08538

⌈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_ERROR.⌋

(RS_E2E_08538)

7.7.4 De-Initialization

[SWS_E2EXf_00148]

Upstream requirements: RS_E2E_08538

⌈E2EXf_DeInit() shall set the module initialization state to FALSE.⌋

7.8 Error classification

7.8.1 Development Errors

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

[SWS_E2EXf_00137] Definiton of development errors in module E2EXf

Upstream requirements: RS_E2E_08538

⌈

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

⌋

[SWS_E2EXf_00144]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00145]

Upstream requirements: RS_E2E_08538

⌈If the XfrmDevErrorDetect switch is enabled and the configuration variant is VARIANT-POST-BUILD, the function E2EXf_Init shall check the contents of the given configuration set for being within the allowed boundaries. If the function E2EXf_Init detects an error, then it shall skip the initialization of the E2E Transformer, keep the module inter-

nal state as uninitialized and it shall report development error code E2EXF_E_INIT_FAILED to the Det_ReportError service of the Default Error Tracer.⌋

[SWS_E2EXf_00146]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00149]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00150]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00151]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00152]

Upstream requirements: RS_E2E_08538

⌈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.⌋

[SWS_E2EXf_00153]

Upstream requirements: RS_E2E_08538

⌈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.⌋

7.8.2 Runtime Errors

[SWS_E2EXf_00122]

Upstream requirements: RS_E2E_08538

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

[SWS_E2EXf_00009]

Upstream requirements: RS_E2E_08538

⌈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).⌋

Note: If there is a hard error in one of the transformers then the transformer chain is to be aborted. The first hard error shall be passed to the server application. If there is at least one soft error then the first soft error shall be passed to the application. reported by E2E Transformer.

7.8.3 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.

7.8.4 Extended Production Errors

All Extended Production Errors valid for E2E Transformer are specified in [2, 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] Definition of imported datatypes of module E2EXf

Upstream requirements: RS_E2E_08538, RS_E2E_08528

⌈

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

▽

△

Module	*Header File*	*Imported Type*
	E2E.h	E2E_P07mProtectStateType
	E2E.h	E2E_P08CheckStateType
	E2E.h	E2E_P08CheckStatusType
	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, [2, ASWS Transformer] defines types which shall be imported.

8.2 Type definitions

8.2.1 E2EXf_ConfigType

[SWS_E2EXf_00030] Definition of datatype E2EXf_ConfigType

Upstream requirements: RS_E2E_08538

⌈

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	

⌋

8.2.2 E2EXf_CSTransactionHandleType

[SWS_E2EXf_00026] Definition of datatype E2EXf_CSTransactionHandleType

Upstream requirements: RS_E2E_08538

⌈

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	

⌋

8.3 Function definitions

8.3.1 E2EXf_<transformerId>

[SWS_E2EXf_00032] Definition of API function E2EXf_<transformerId>

Upstream requirements: RS_E2E_08538

⌈

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	

⌋

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

8.3.2 E2EXf_Inv_<transformerId>

[SWS_E2EXf_00034] Definition of API function E2EXf_Inv_<transformerId>

Upstream requirements: RS_E2E_08538

⌈

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 if disableEndToEndStateMachine equals FALSE. The high nibble shall be set to 0x6 if the E2E state machine is disabled (disable EndToEndStateMachine equals TRUE). 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) 0x60 (E_SAFETY_NOSM_OK) 0x61 (E_SAFETY_NOSM_REP) 0x62 (E_SAFETY_NOSM_SEQ) 0x63 (E_SAFETY_NOSM_ERR) 0x65 (E_SAFETY_NOSM_NND) 0x66 (E_SAFETY_NOSM_DEC) ▽

▽

△

		△ 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. 0x8D (E_E2E_HARD_SAFETY_ERR): A runtime error occured, during Client/Server-Communication on the server side, safety properties could not be checked and no output data could be produced (see [SWS_E2EXf_00167]). 0xFF (E_SAFETY_HARD_RUNTIMEERROR): A runtime error occured, safety properties could not be checked and no output data could be produced.
Description	Checks the received data. If the data can be used by the caller, then the function returns E_OK.	
Available via	E2EXf.h	

⌋

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

In case the state machine is disabled: If E2E_PXXMapStatusToSM (after E2E_PXXCheck() call) returns E2E_P_OK the function returns E_SAFETY_NOSM_OK. Technically E_SAFETY_NOSM_OK is implemented as a specific soft error (value between 0x01 and 0x7F). The application should evaluate E_SAFETY_NOSM_OK not as an error.

8.3.3 E2EXf_Init

Add the following function:

[SWS_E2EXf_00035] Definition of API function E2EXf_Init

Upstream requirements: RS_E2E_08538

⌈

Service Name	E2EXf_Init	

Syntax	`void E2EXf_Init (`
	`  const E2EXf_ConfigType* config`
	`)`	

Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non 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

⌋

8.3.4 E2EXf_DeInit

[SWS_E2EXf_00138] Definition of API function E2EXf_DeInit

Upstream requirements: RS_E2E_08538

⌈

Service Name	E2EXf_DeInit

Syntax	`void E2EXf_DeInit (`
	`  void`
	`)`

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

⌋

8.3.5 E2EXf_GetVersionInfo

[SWS_E2EXf_00036] Definition of API function E2EXf_GetVersionInfo

Upstream requirements: RS_E2E_08538

⌈

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	

⌋

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] Definition of mandatory interfaces required by module E2EXf

Upstream requirements: RS_E2E_08538

⌈

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

▽

△

API Function	*Header File*	*Description*
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.

⌋

8.6.2 Optional Interfaces

None

8.6.3 Configurable interfaces

None

9 Sequence diagrams

The following chapter contains sequence diagrams to show the interaction of AUTOSAR modules for E2E protected communication and the specifics per communication type - signal based communication, service oriented communication, signal to service translation (see Chapter 7).

9.1 E2E for Sender/Receiver

9.1.1 Send E2E protected signals

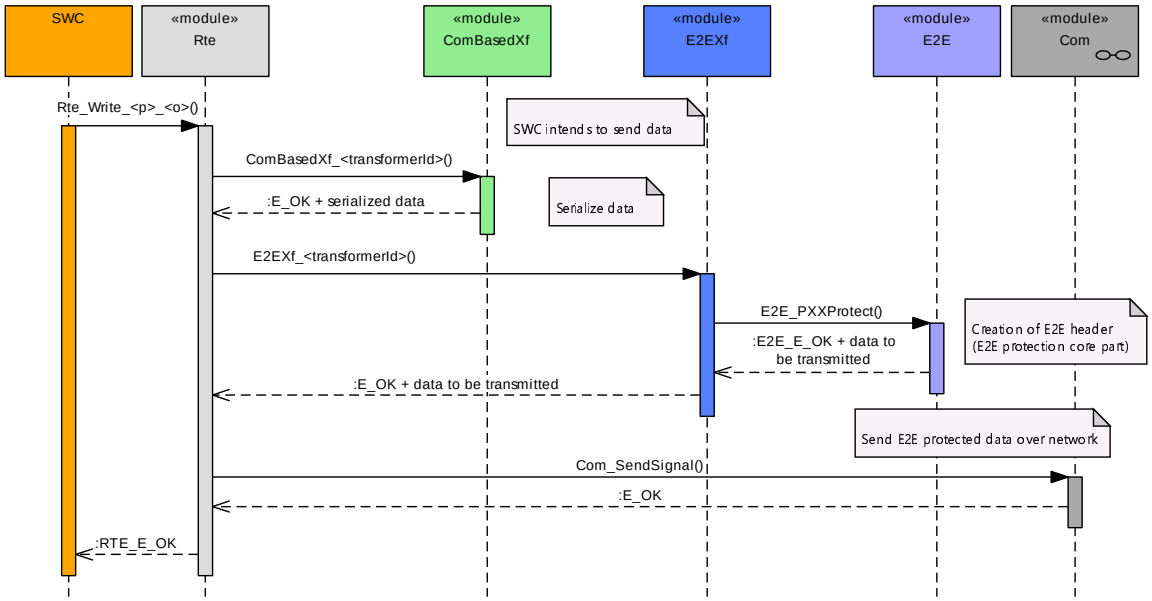

Figure 9.1: E2E protect sequence for signal based communication (E2EXf_<transformerId>)

9.1.2 Receive E2E protected signals (Activation mode)

The following sequence diagram shows how sender/receiver communication for data transmission and activation of runnable entity (activation mode) on the receiver side may be implemented (see figure "Sender Receiver communication with event semantics and activation of runnable entity as reception mechanism" in [10, CP-SWS-RTE]).

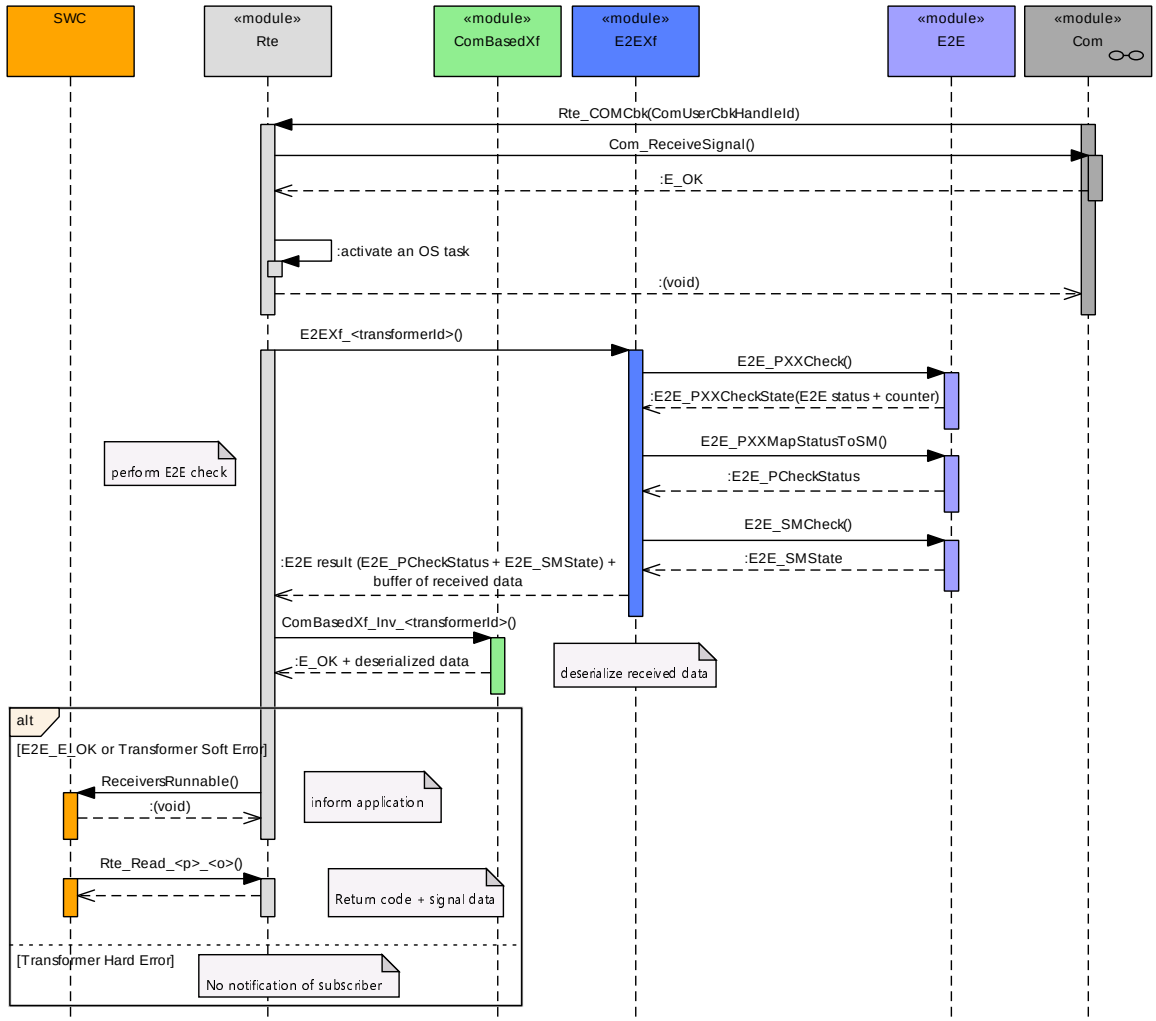

Figure 9.2: E2E check sequence for signal based communication - activation mode (E2EXf_Inv<transformerId>)

9.1.3 Receive E2E protected signals (Polling mode)

The following sequence diagram shows how sender/receiver communication for data transmission in polling mode on the receiver side may be implemented (see "Sender Receiver communication with data semantics and dataReceive- PointByArgument as reception mechanism" in [10, CP-SWS-RTE]).

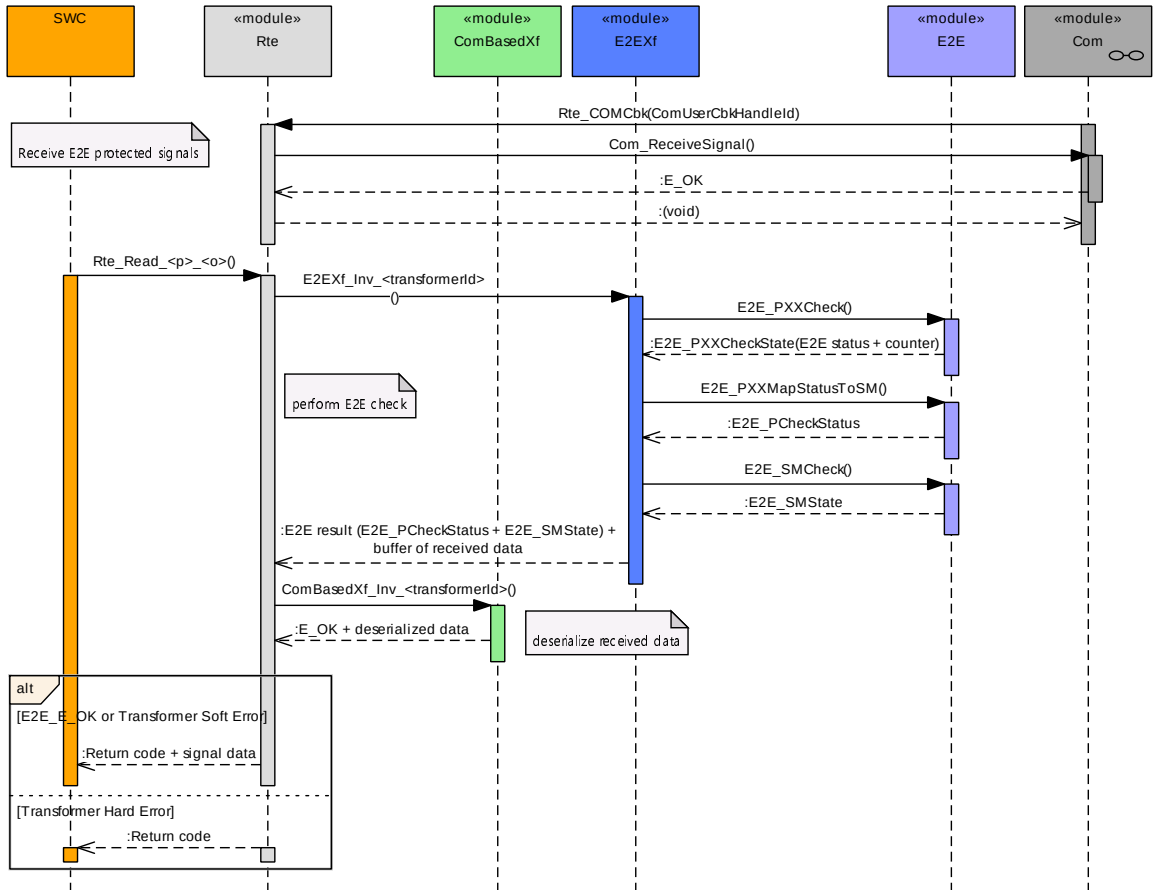

Figure 9.3: E2E check sequence for signal based communication - polling mode (E2EXf_Inv<transformerId>)

9.2 E2E for Events

The following sequence shows the publisher/subscriber pattern. When an event is updated all subscribers receive an update message. RTE calls SomeIp transformer (serialization) and E2E transformer for E2E protection..

9.2.1 Send an E2E Protected Event

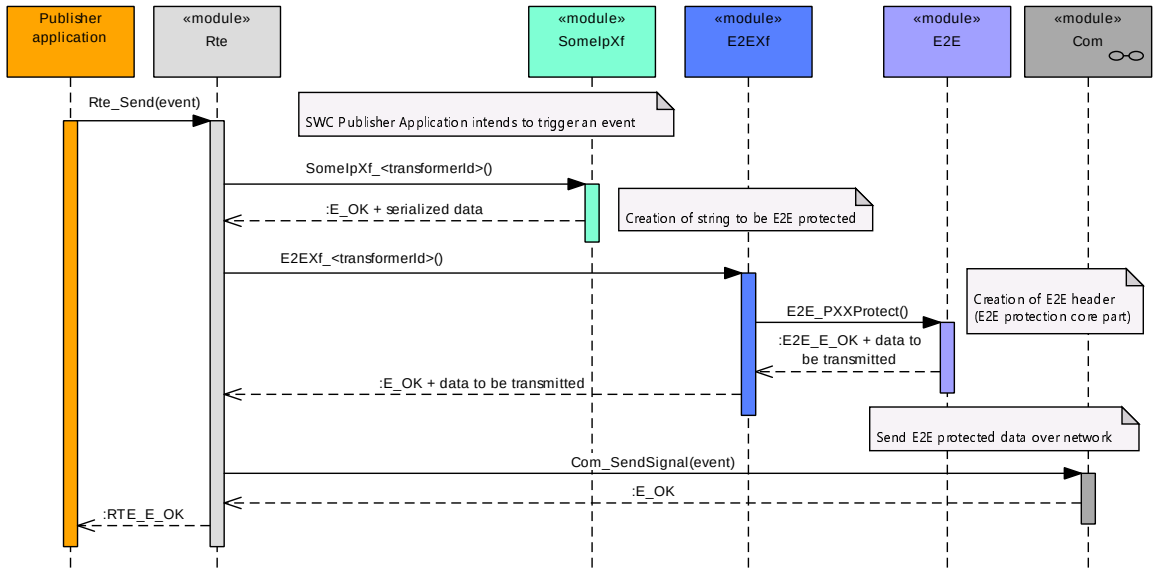

Figure 9.4: E2E protect sequence for events (E2EXf_<transformerId>)

9.2.2 Receive an E2E Protected Event(Activation Mode)

The following sequence diagram shows publisher/subscriber communication for data transmission and activation of runnable entity (activation mode) on the subscriber side may be implemented (see figure "Sender Receiver communication with event semantics and activation of runnable entity as reception mechanism" in [10, CP-SWS-RTE]).

In this mode, the receipt of messages triggers the RTE to call the subscriber's application.

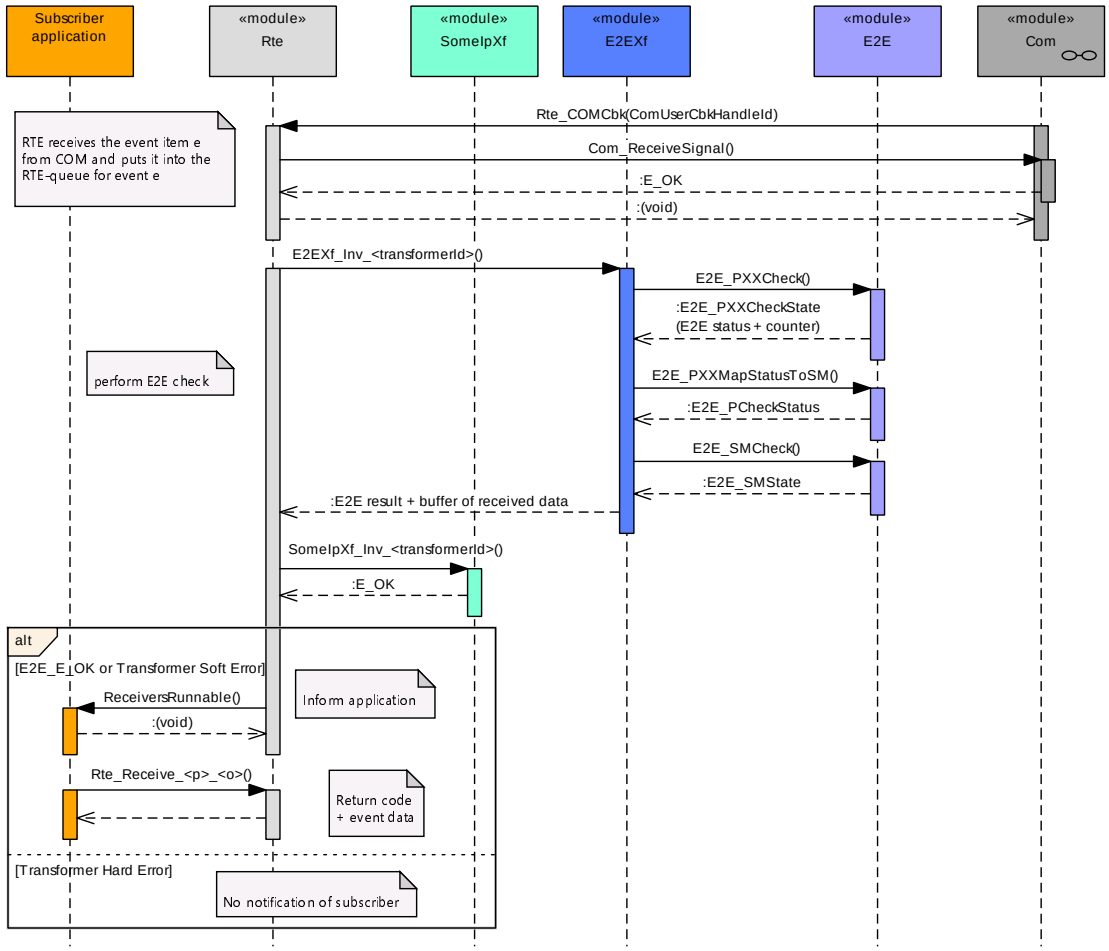

Figure 9.5: E2E check sequence for events - activation mode (E2EXf_<transformerId>)

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 Transformer for deserialization if there is no E2E error. In case of E2E error the subscriber is not updated, in case of no E2E error the subscriber gets the updated value of the event.

Inverse E2E transformer and the inverse SOME/IP transformer run in sequence. The behavior of transformers in a chain and the possible errors are described in chapter "Error Handling" of [2, ASWS Transformer General].

9.2.3 Receive an E2E Protected Event(Polling Mode)

The following sequence diagram shows publisher/subscriber communication for data transmission in polling mode on the subscriber side may be implemented (see figure "Sender Receiver communication with event semantics and dataReceive- PointByArgument as reception mechanism" in [10, CP-SWS-RTE]).

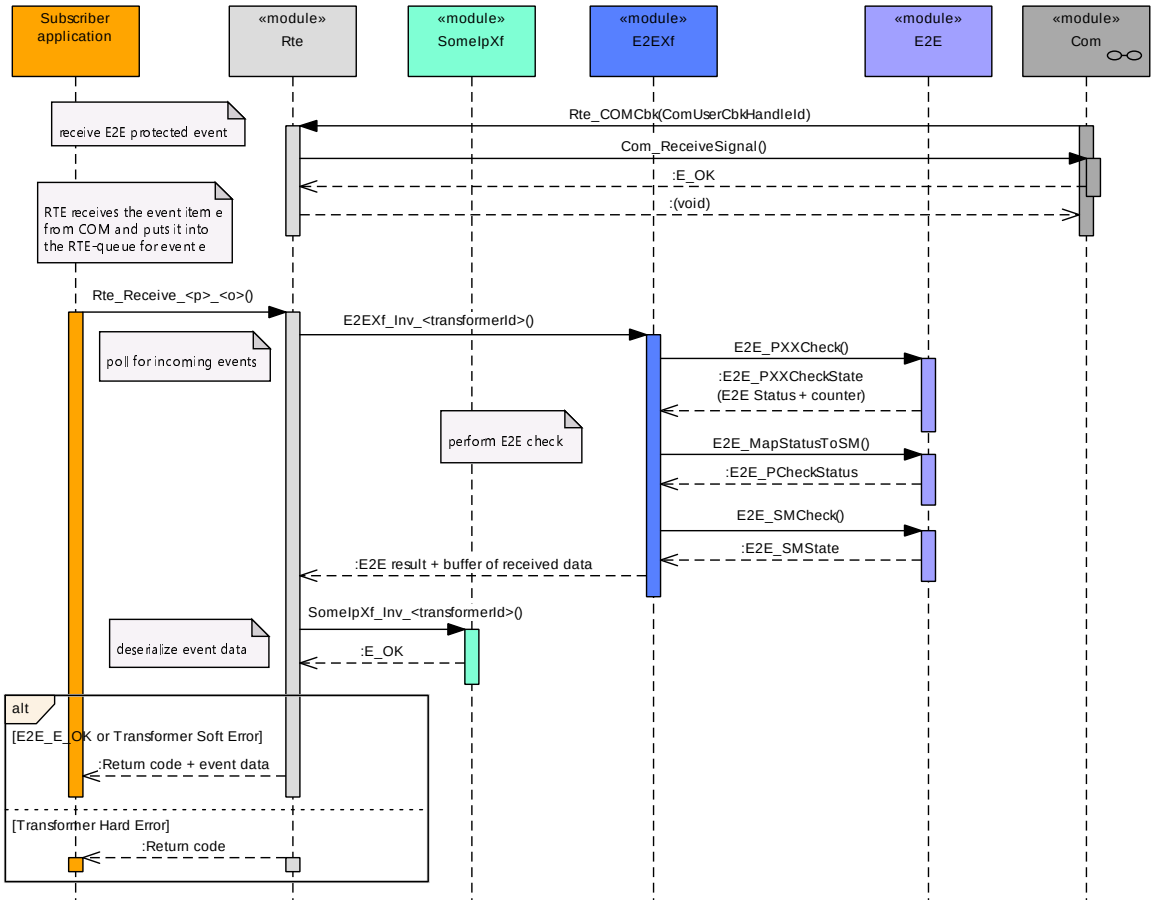

Figure 9.6: E2E check sequence for events - polling mode (E2EXf_<transformerId>)

9.3 E2E for Method Call/Method Response

The following sequences show the method call/method response pattern.

9.3.1 Call an E2E Protected Method

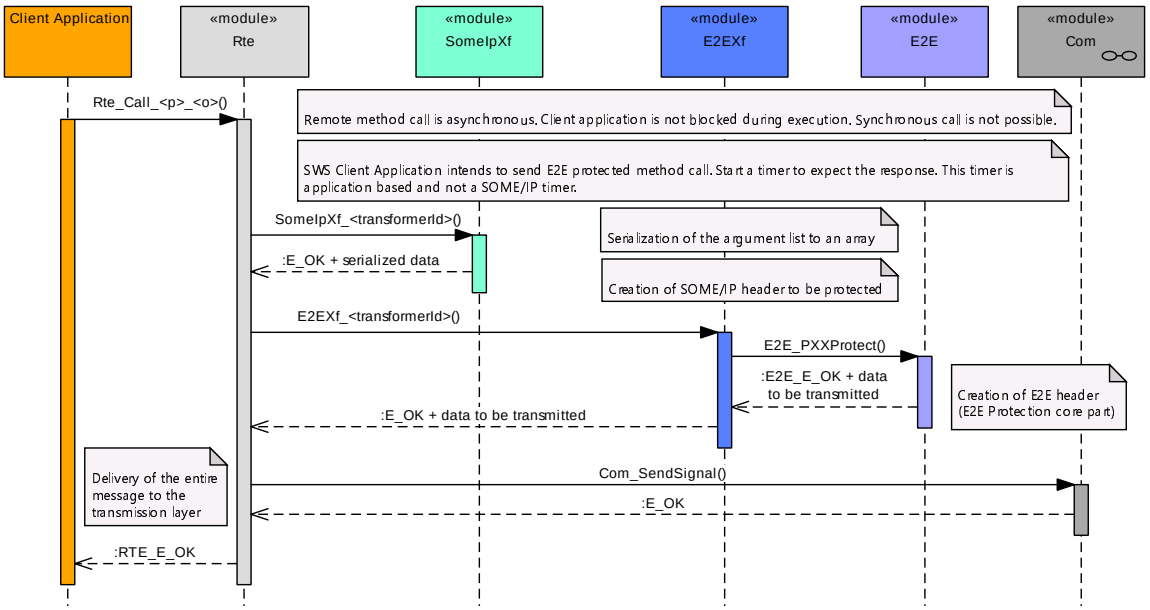

Figure 9.7: E2E protect sequence for methods (E2EXf_<transformerId>)

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 SomeIp 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 and respond to an E2E Protected Method Call

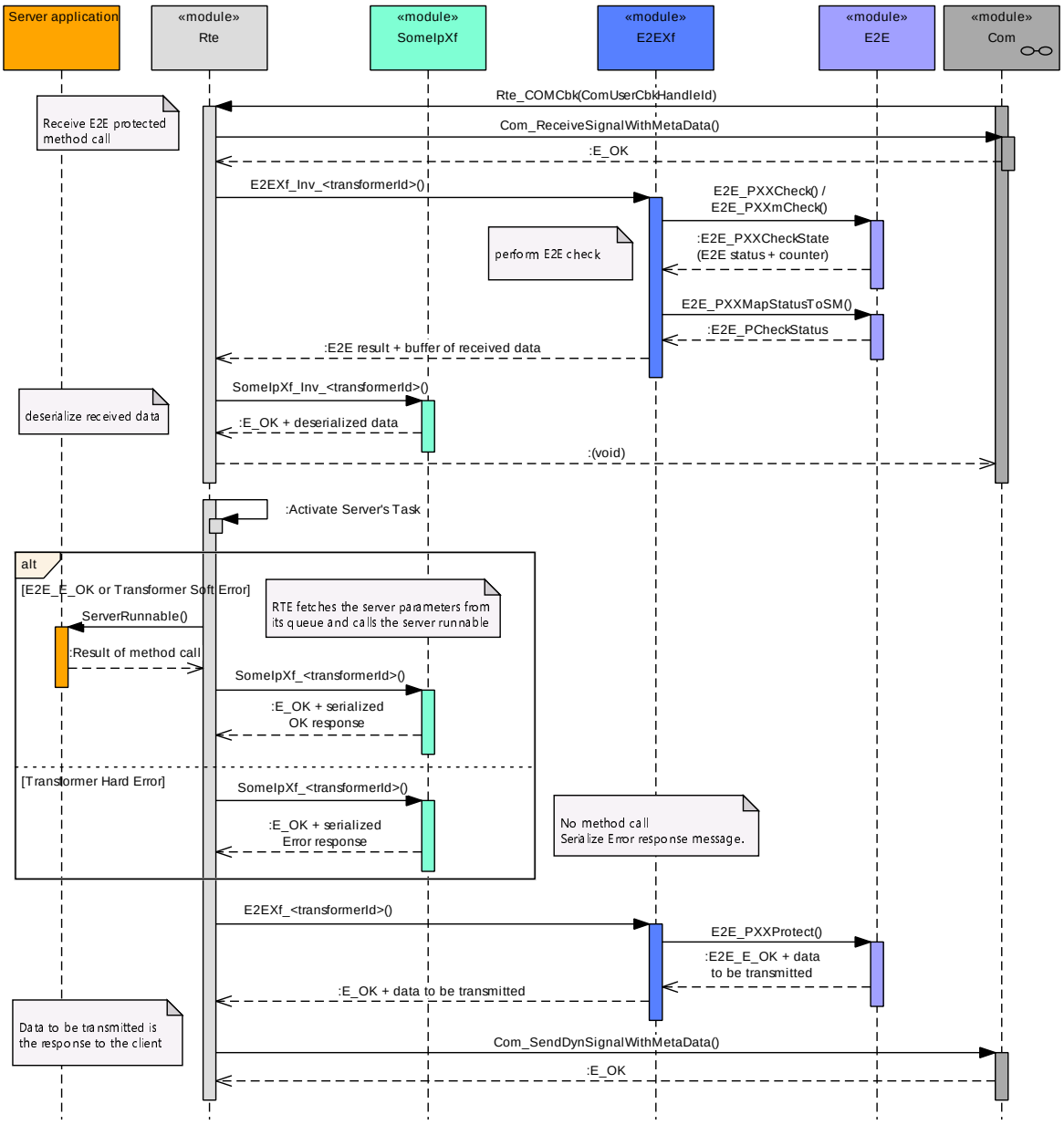

Figure 9.8: E2E check sequence for method request (E2EXf_Inv<transformerId>)

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/error message is set up and also serialized and E2E protected before it is sent back.

9.3.3 Receive a E2E Protected Response to a Method Call

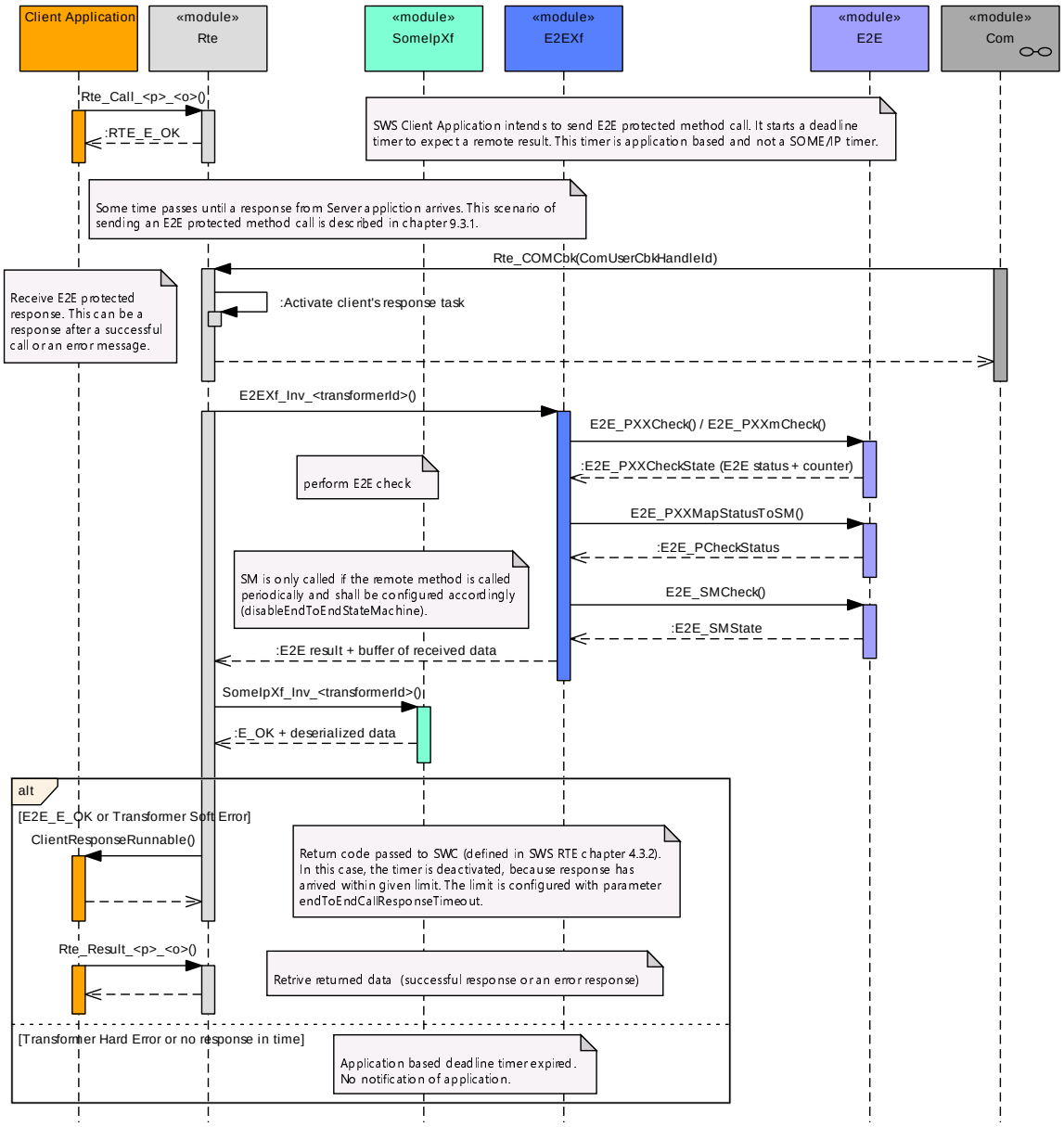

Figure 9.9: E2E check sequence for method reponse (E2EXf_Inv<transformerId>)

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 an error message 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 provided as response message.

9.4 E2E in Signal to Service Translation

The following sequence shows signal to service translation with E2E protection.

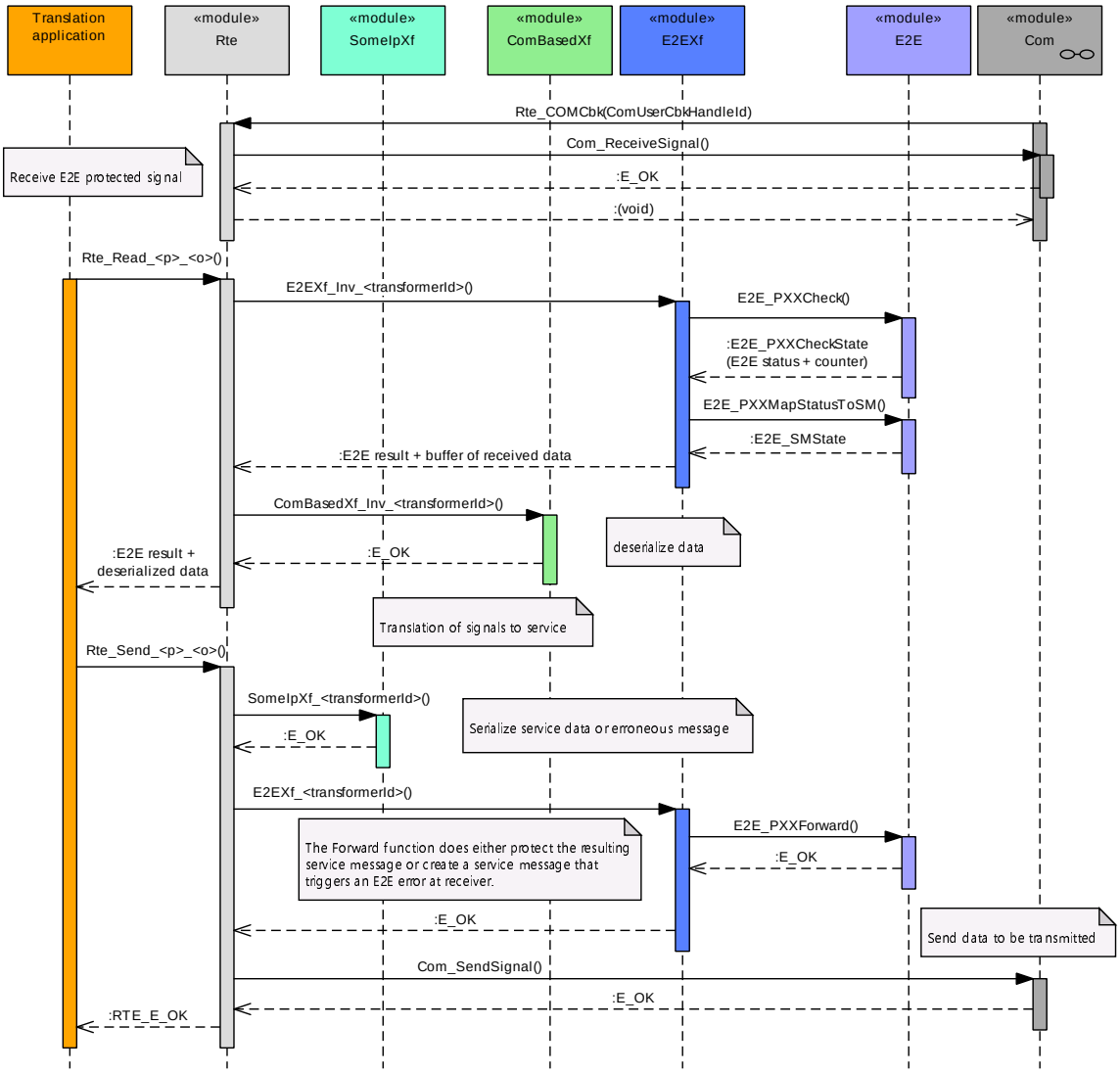

Figure 9.10: E2E sequence for signal to service translation (E2EXf_<TransformerId>)

The Translation Application creates a new message which is built out of the deserialized data and the E2E result. If this E2E result is an error then the new message is built up that way that it will also call an E2E error at its receiver.

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]

Upstream requirements: SRS_BSW_00159

⌈The apiServicePrefix of the E2E Transformer's EcuC shall be set to E2EXf.⌋

A Not applicable requirements

[SWS_E2EXf_NA_00001]

Upstream requirements: RS_E2E_08544

⌈These requirements are not applicable to this specification.⌋

B Change History

B.1 Change History R22-11

B.2 Change History R23-11

B.2.1 Added Specification Items in R23-11

none

B.2.2 Changed Specification Items in R23-11

[SWS_E2EXf_00026] [SWS_E2EXf_00028] [SWS_E2EXf_00030] [SWS_E2EXf_00032] [SWS_E2EXf_00034] [SWS_E2EXf_00037] [SWS_E2EXf_00047] [SWS_E2EXf_00087] [SWS_E2EXf_00168] [SWS_E2EXf_00175] [SWS_E2EXf_00208] [SWS_E2EXf_NA_00001]

B.2.3 Deleted Specification Items in R23-11

[SWS_E2EXf_00120] [SWS_E2EXf_00163] [SWS_E2EXf_00176] [SWS_E2EXf_00177] [SWS_E2EXf_00178] [SWS_E2EXf_00179] [SWS_E2EXf_00204] [SWS_E2EXf_00205]

B.3 Change History R24-11

B.3.1 Added Specification Items in R24-11

none

B.3.2 Changed Specification Items in R24-11

[SWS_E2EXf_00034] [SWS_E2EXf_00035] [SWS_E2EXf_00088] [SWS_E2EXf_00138]

B.3.3 Deleted Specification Items in R24-11

none