

Document Title	Specification of SW-C End-to-End Communication Protection Library
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	428
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R20-11

Document Change History			
Date	Release	Changed by	Change Description
2020-11-30	R20-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • New profiles 4m,7m,08,44 • E2E for methods • Extension of E2E State Machine
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> • Incorporated E2E_PxxForward methods to replicate detected E2E-Errors on outgoing messages • E2E_P0xSTATUS_ERROR values are now the same for all profiles • Fixed minor inconsistencies and typos • Updated Tracing from SRS_E2E to RS_E2E • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> • Added clarification regarding assumptions on failure modes and detection capabilities in annex A. • Fixed inconsistent definition of length in E2E header for P04, P05, and P06 • Clarification of parameters CounterOffset and CRCOffset in E2E_P01ConfigType

Document Change History			
Date	Release	Changed by	Change Description
2017-12-08	4.3.1	AUTOSAR Release Management	<ul style="list-style-type: none"> Updated traceability to SRS E2E. Fixed enumeration literals for E2E_PxxCheckStatusType for profiles 1 and 2. Corrected name of step E2E_SMClearProfileStatus to E2E_SMClearStatus in Routine E2E_SM_checkinit Various clarifications in configuration and routine parameters, mainly of profile 2 and 7.
2016-11-30	4.3.0	AUTOSAR Release Management	<ul style="list-style-type: none"> Added new Profiles 7, 11 and 22. Fixed initialization of profile 1 and 2 in the init function. Now properly sets WaitForFirstData to TRUE. Corrected/unified initialization of Counter state variable and bit/byte conversion in configuration data in profiles 4, 5, and 6. Removed chapter 8.3.7 elementary protocol functions that were marked obsolete since several releases.
2015-07-31	4.2.2	AUTOSAR Release Management	<ul style="list-style-type: none"> Introduced new E2E state machine profile status E2E_P_NONEWDATA. Adapted figures, API tables and mapping functions. This solves an issue with deterministic startup of the state machine. Updated Figure 7-7, added behavior in case ReceivedCounter is out of range. Assigned new specification ID SWS_E2E_00478 to duplicate specification SWS_E2E_00324 (specification of profile 4). Fixed figure 7-6 “Calculate CRC over Data ID and Data”, which was already fixed in R4.1.2 but falsely included as of R4.1.1.

Document Change History			
Date	Release	Changed by	Change Description
2014-10-31	4.2.1	AUTOSAR Release Management	<ul style="list-style-type: none"> • Introduction of E2E profiles 4, 5, 6 • Introduction of E2E state machine • Introduction of init functions and status mapping functions for profiles 1, 2 • Overview of wrapper, by means of several new diagrams.
2014-03-31	4.1.3	AUTOSAR Release Management	<ul style="list-style-type: none"> • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	<ul style="list-style-type: none"> • Correction in E2E variant 1C • Various minor corrections • Editorial changes
2013-03-15	4.1.1	AUTOSAR Release Management	<ul style="list-style-type: none"> • Full support for E2E protection at signal group level • Removed dependency to Rte_IsUpdated • Changed recommendations about the maximum data lengths • Addition of initialization functions to the redundant wrapper • Corrections in code examples
2013-03-15	4.1.1	AUTOSAR Administration	<ul style="list-style-type: none"> • Reworked according to the new SWS_BSWGeneral • New indexing scheme for requirements • Extension of E2E Profile 1 to support 12-bit Data IDs (variant 1C) • Alignment with ISO 26262 (terms, communication faults) • Quality ameliorations (due to document review) • Clarification in the configuration of E2E parameters

Document Change History			
Date	Release	Changed by	Change Description
2011-12-22	4.0.3	AUTOSAR Administration	<ul style="list-style-type: none"> • E2E Profile 3 removed (not backward compatible) • Several bugfixes in of E2E Protection Wrapper API (not backward compatible) • Modified return values of E2E Protection Wrapper API (not backward compatible) • Addition of init API for the E2E Protection Wrapper • Several bugfixes and modifications in code examples of E2E Protection Wrapper • Extensions in configuration, making sender and receiver more independent • Bugfix in the profile 1 alternating mode CRC calculation • Clarifications with in E2E Profile 1 with respect to the CRC • Several minor bug fixes • Several optimizations in the text descriptions • New template with requirements traceability
2010-09-30	3.1.5	AUTOSAR Administration	<ul style="list-style-type: none"> • Corrected the wrapper configuration. • Corrected the code example for the usage of the wrapper.
2010-02-02	3.1.4	AUTOSAR Administration	<ul style="list-style-type: none"> • Initial Release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1	Introduction and functional overview	9
2	Acronyms and abbreviations.....	11
3	Related documentation	12
3.1	Input documents.....	12
3.2	Related standards and norms.....	12
4	Constraints and assumptions.....	13
4.1	Limitations	13
4.2	Implementation of the E2E Library	13
5	Dependencies to/from other modules.....	14
5.1.1	Required file structure	14
5.1.2	Dependency on CRC library	14
6	Requirements traceability.....	15
7	Functional specification.....	29
7.1	Error classification.....	29
7.1.1	Development Errors	30
7.1.2	Runtime Errors	30
7.1.3	Transient Faults.....	30
7.1.4	Production Errors.....	30
7.1.5	Extended Production Errors	30
8	API specification.....	31
8.1	Imported types.....	31
8.2	Type definitions	31
8.2.1	E2E Profile 1 types.....	32
8.2.2	E2E Profile 2 types.....	37
8.2.3	E2E Profile 4 types.....	42
8.2.4	E2E Profile 4m types.....	45
8.2.5	E2E Profile 5 types.....	49
8.2.6	E2E Profile 6 types.....	53
8.2.7	E2E Profile 7 types.....	57
8.2.8	E2E Profile 7m types.....	61
8.2.9	E2E Profile 8 types.....	64
8.2.10	E2E Profile 11 types.....	68
8.2.11	E2E Profile 22 types.....	72
8.2.12	E2E Profile 44 types.....	76
8.2.13	E2E state machine types.....	80
8.3	Routine definitions.....	84
8.3.1	E2E Profile 1 routines.....	84
8.3.2	E2E Profile 2 routines.....	89
8.3.3	E2E Profile 4 routines.....	95
8.3.4	E2E Profile 4m routines.....	99
8.3.5	E2E Profile 5 routines.....	106
8.3.6	E2E Profile 6 routines.....	111
8.3.7	E2E Profile 7 routines.....	115
8.3.8	E2E Profile 7m routines.....	121

8.3.9	E2E Profile 8 routines.....	128
8.3.10	E2E Profile 11 routines.....	133
8.3.11	E2E Profile 22 routines.....	138
8.3.12	E2E Profile 44 routines.....	143
8.3.13	E2E State machine routines.....	148
8.3.14	Auxiliary Functions	150
8.4	Call-back notifications	150
8.5	Scheduled functions.....	150
8.6	Expected Interfaces	151
8.6.1	Mandatory Interfaces.....	151
9	Sequence Diagrams for invoking E2E Library.....	152
9.1	Sender	152
9.1.1	Sender of data elements	152
9.1.2	Sender at signal group level.....	154
9.2	Receiver	155
9.2.1	Receiver at data element level.....	157
9.2.2	Receiver at signal group level	159
10	Configuration specification.....	162
10.1	Published Information	162
11	Annex A: Safety Manual for usage of E2E Library	163
11.1	E2E profiles and their standard variants.....	163
11.2	E2E error handling	163
11.3	Methodology of usage of E2E Library.....	163
11.4	RTE configuration constraints for SW-C level protection	164
11.4.1	Communication model for SW-C level protection	164
11.4.2	Multiplicities for SW-C level protection.....	164
11.4.3	Explicit access	164
11.5	Restrictions on the use of COM features.....	165
12	Annex B: Application hints on usage of E2E Library	167
12.1	E2E Protection Wrapper	168
12.1.1	Functional overview.....	169
12.1.2	Application scenario with Transmission Manager.....	170
12.1.3	Application scenario with E2E Manager and Conversion Manager	172
12.1.4	File structure.....	175
12.1.5	Methodology	176
12.1.6	Error classification	180
12.1.7	E2E Protection Wrapper routines.....	182
12.1.8	E2EPW Routines Diagrams	199
12.1.9	Code Example	207
12.2	COM E2E Callouts	225
12.2.1	Functional overview.....	226
12.2.2	Methodology	229
12.2.3	Code Example	232
12.3	Provision of the Protection Wrapper Interface on a ECU with COM Callout solution	233
12.4	Protection at RTE level through E2E Transformer	233
13	Not applicable requirements	234

1 Introduction and functional overview

This document contains the platform specific implementation requirements of the PRS E2E Protocol. This includes interfaces and datatypes used.

The main part of the functional specification is given in the AUTOSAR Foundation document 849 “E2E Protocol Specification”.

Platform dependent functional specifications extending the protocol specifications are collected in the following sub section(s).

The concept of E2E protection assumes that safety-related data exchange shall be protected at runtime against the effects of faults within the communication link (see Figure 1-1). Examples for such faults are random HW faults (e.g. corrupt registers of a CAN transceiver), interference (e.g. due to EMC), and systematic faults within the software implementing the VFB communication (e.g. RTE, IOC, COM and network stacks).

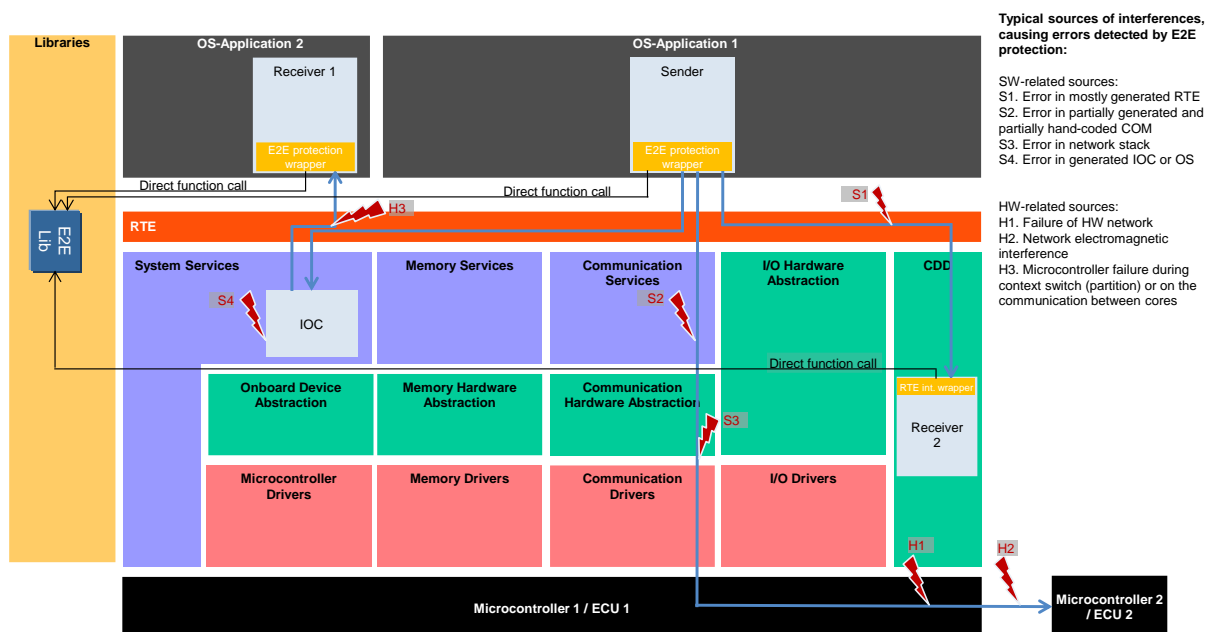


Figure 1-1: Example of faults mitigated by E2E protection

By using E2E communication protection mechanisms, the faults in the communication link can be detected and handled at runtime. The E2E Library provides mechanisms for E2E protection, adequate for safety-related communication having requirements up to ASIL D.

The algorithms of protection mechanisms are implemented in the E2E Library. The callers of the E2E Library are responsible for the correct usage of the library, in particular for providing correct parameters the E2E Library routines.

The E2E protection allows the following:

1. It protects the safety-related data elements to be sent over the RTE by attaching control data,
2. It verifies the safety-related data elements received from the RTE using this control data, and

3. It indicates that received safety-related data elements faulty, which then has to be handled by the receiver SW-C.

To provide the appropriate solution addressing flexibility and standardization, AUTOSAR specifies a set of flexible E2E profiles that implement an appropriate combination of E2E protection mechanisms. Each specified E2E profile has a fixed behavior, but it has some configuration options by function parameters (e.g. the location of CRC in relation to the data, which are to be protected).

The E2E library is invoked from:

1. E2E Transformer (a new, standardized way to invoke E2E, introduced in R4.2.1)
2. E2E Protection Wrapper
3. COM E2E Callout.

Regardless where E2E is executed, the E2E Protection is for data elements. The E2E Protection is performed on the serialized representation of data elements, on the same bit layout as the one transmitted on the bus. This means:

1. In case E2E Transformer is used, the serialization is performed by a transformer above E2E Transformer (COM-based transformer or Some/IP transformer).
2. In case E2E Protection Wrapper is used, the wrapper needs to serialize the data element into the serialized form of the corresponding signal group (in other words, the wrapper creates a part of I-PDU that represents the signal group and at the same time the data element).
3. In case the COM callout is used, the serialization is done by the communication stack (RTE, COM), so the callout operates directly on the serialized signal groups in the I-PDU.

A data element (and the corresponding signal group) is either completely E2E-protected, or it is not protected. It is not possible to protect a part of it.

An I-PDU may carry several data elements (and corresponding signal groups). It is possible to independently E2E-protect a subset of these data elements.

An appropriate usage of the E2E Library alone is not sufficient to achieve a safe E2E communication according to ASIL D requirements. Solely the user is responsible to demonstrate that the selected profile provides sufficient error detection capabilities for the considered network (e.g. by evaluation hardware failure rates, bit error rates, number of nodes in the network, repetition rate of messages and the usage of a gateway).

2 Acronyms and abbreviations

All technical terms used in this document, except the ones listed in the table below, can be found in the official AUTOSAR glossary [7].

Acronyms and abbreviations that have a local scope and therefore are not contained in the AUTOSAR glossary appear in the glossary below.

Abbreviation / Acronym:	Description:
E2E Library	Short name for the End-to-End Communication Protection Library.
Data ID	An identifier that uniquely identifies the message / data element / data.
Source ID	An identifier that uniquely identified the source (origin) of the message / data element / data.
Repetition	Repetition of information.
Loss	Loss of information.
Delay	Delay of information.
Insertion	Insertion of information.
Masquerade	Masquerade.
Incorrect addressing	Incorrect addressing of information.
Incorrect sequence	Incorrect sequence of information.
Corruption	Corruption of information.
Asymmetric information	Asymmetric information sent from a sender to multiple receivers.
Subset	Information from a sender received by only a subset of the receivers.
Blocking	Blocking access to a communication channel.

Table 2-1: Acronyms and abbreviations

In the whole document, there are many requirements that apply to all E2E Profiles at the same time. Such requirements are defined as one requirement that applies to all profiles at the same time. In case some names are profile dependent, then XX notation is used: if in a requirement appears the string containing XX, then it is developed to strings with 01, 02, 04, 4m, 05, 06, 07, 7m, 11, 12, 22, and 44 respectively instead of XX. For example, E2E_PXXCheck() develops to the following two E2E_P01Check(), E2E_P02Check() a.s.o.

3 Related documentation

3.1 Input documents

- [1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf
- [2] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf
- [3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf
- [4] Specification of CRC Routines
AUTOSAR_SWS_CRCLibrary.pdf
- [5] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf
- [6] Requirements on Libraries
AUTOSAR_SRS_Libraries.pdf
- [7] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf
- [8] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf
- [9] System Template
AUTOSAR_TPS_SystemTemplate.pdf
- [10] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf
- [11] E2E Protocol Specification
AUTOSAR_PRS_E2EProtocol.pdf

3.2 Related standards and norms

- [12] ISO 26262-6:2018
<http://www.iso.org/>

4 Constraints and assumptions

4.1 Limitations

General limitations regarding E2E protection and the detectable failure modes are described in [11]. E2E Profile 2 has in R4.2.1 a new setting offset. This offset can be configured in the system template. However, the E2E Profile 2 specification does not support the case when offset is different than 0. The specification of E2E Profile 2 will be fixed in a future AUTOSAR release, to support a configurable offset.

E2E Profile 1 in the “Double Data ID configuration” uses an implicit 2-byte Data ID, over which CRC8 is calculated. As a CRC over two different 2-byte numbers may result with the same CRC, some precautions must be taken by the user. See [UC E2E_00072](#) and [UC E2E_00073](#).

E2E Profile 2 uses an implicit 1-byte Data ID, selected from a List of Data IDs depending on each value of the counter, for calculation of the CRC. See chapter 13 for details on the usage and generation of DataIDList for E2E profile 2.

4.2 Implementation of the E2E Library

[SWS_E2E_00050]「The implementation of the E2E Library shall comply with the requirements for the development of safety-related software for the automotive domain.」
(RS_E2E_08527)

The ASIL assigned to the requirements implemented by the E2E library depends on the safety concept of a particular system. Depending on that application, the E2E Library at least may need to comply with an ASIL A, B, C or D development process. Therefore, it may be most efficient to develop the library according to the highest ASIL, which enables to use the same library for lower ASILs as well.

[SWS_E2E_00311]「The configuration of the E2E Library and of the code invoking it (e.g. E2E wrapper , E2E callouts , E2E transformer) shall be implemented and configured (including configuration options used from other subsystems, e.g. COM signal to I-PDU mapping) according to the requirements for the development of safety-related software for the automotive domain.」(RS_E2E_08528)

5 Dependencies to/from other modules

5.1.1 Required file structure

[SWS_E2E_00048] E2E library shall be built of the following files: E2E.h (common header), E2E.c (implementation of common parts), E2E_PXX.c (where XX: e.g. 01, 02, ... representing the profile) and E2E_SM.c (for E2E state machine). (RS_E2E_08528)

[SWS_E2E_00215] Files E2E_PXX.c and E2E.h shall contain implementation parts specific of each profile. (RS_E2E_08528)

The below requirement is redundant with above ones, but important to be stated explicitly:

[SWS_E2E_00115] E2E library files (i.e. E2E_*.*) shall not include any RTE files. (RS_E2E_08528)

5.1.2 Dependency on CRC library

It is important to note that the function Crc_CalculateCRC8 of CRC library / CRC routines have changed its functionality since R4.0, i.e. it is different in R3.2 and >=R4.0:

1. There is an additional parameter Crc_IsFirstCall
2. The function has different start value and different XOR values (changed from 0x00 to 0xFF).

This results with a different value of computed CRC of a given buffer.

To have the same results of the functions E2E_P01Protect() and E2E_P01Check() in >=R4.0 and R3.2, while using differently functioning CRC library, E2E “compensates” different behavior of the CRC library. This results with different invocation of the CRC library by E2E library in >=R4.0 and R3.2.

6 Requirements traceability

Requirement	Description	Satisfied by
RS_E2E_08527	Implementation of E2E protocol shall fulfill ISO 26262	SWS_E2E_00050, SWS_E2E_00158, SWS_E2E_00160, SWS_E2E_00161, SWS_E2E_00166, SWS_E2E_00338, SWS_E2E_00339, SWS_E2E_00349, SWS_E2E_00350, SWS_E2E_00373, SWS_E2E_00379, SWS_E2E_00382, SWS_E2E_00385, SWS_E2E_00387, SWS_E2E_00390, SWS_E2E_00391, SWS_E2E_00393, SWS_E2E_00446, SWS_E2E_00447, SWS_E2E_00449, SWS_E2E_00450, SWS_E2E_00452, SWS_E2E_00455, SWS_E2E_00457, SWS_E2E_00458, SWS_E2E_00460, SWS_E2E_00546, SWS_E2E_00547, SWS_E2E_00548, SWS_E2E_00549, SWS_E2E_00550, SWS_E2E_00572, SWS_E2E_00573, SWS_E2E_00574, SWS_E2E_00575, SWS_E2E_00576, SWS_E2E_00577, SWS_E2E_00578, SWS_E2E_00579, SWS_E2E_00580, SWS_E2E_00581, SWS_E2E_00583, SWS_E2E_00584, SWS_E2E_00585, SWS_E2E_00586, SWS_E2E_00587, SWS_E2E_00588, SWS_E2E_00589, SWS_E2E_00590, SWS_E2E_91001, SWS_E2E_91002, SWS_E2E_91003, SWS_E2E_91004, SWS_E2E_91005, SWS_E2E_91006, SWS_E2E_91012, SWS_E2E_91013, SWS_E2E_91014, SWS_E2E_91015, SWS_E2E_91017, SWS_E2E_91018, UC_E2E_00304, UC_E2E_00328
RS_E2E_08528	E2E protocol shall provide different E2E profiles	SWS_E2E_00011, SWS_E2E_00017, SWS_E2E_00018, SWS_E2E_00020, SWS_E2E_00021, SWS_E2E_00033, SWS_E2E_00048, SWS_E2E_00110, SWS_E2E_00115, SWS_E2E_00152, SWS_E2E_00153, SWS_E2E_00154, SWS_E2E_00158, SWS_E2E_00160, SWS_E2E_00161, SWS_E2E_00166, SWS_E2E_00200, SWS_E2E_00215, SWS_E2E_00311, SWS_E2E_00314, SWS_E2E_00318, SWS_E2E_00319, SWS_E2E_00320, SWS_E2E_00321, SWS_E2E_00322, SWS_E2E_00323, SWS_E2E_00324, SWS_E2E_00325, SWS_E2E_00334, SWS_E2E_00335, SWS_E2E_00336, SWS_E2E_00337, SWS_E2E_00338, SWS_E2E_00339, SWS_E2E_00349, SWS_E2E_00350, SWS_E2E_00353, SWS_E2E_00370, SWS_E2E_00371, SWS_E2E_00373, SWS_E2E_00378, SWS_E2E_00379, SWS_E2E_00380, SWS_E2E_00381,

		<p>SWS_E2E_00382, SWS_E2E_00383, SWS_E2E_00384, SWS_E2E_00386, SWS_E2E_00387, SWS_E2E_00388, SWS_E2E_00389, SWS_E2E_00390, SWS_E2E_00391, SWS_E2E_00392, SWS_E2E_00393, SWS_E2E_00437, SWS_E2E_00438, SWS_E2E_00439, SWS_E2E_00440, SWS_E2E_00441, SWS_E2E_00443, SWS_E2E_00444, SWS_E2E_00445, SWS_E2E_00446, SWS_E2E_00447, SWS_E2E_00449, SWS_E2E_00450, SWS_E2E_00451, SWS_E2E_00452, SWS_E2E_00455, SWS_E2E_00456, SWS_E2E_00457, SWS_E2E_00458, SWS_E2E_00459, SWS_E2E_00460, SWS_E2E_00476, SWS_E2E_00477, SWS_E2E_00542, SWS_E2E_00544, SWS_E2E_00545, SWS_E2E_00546, SWS_E2E_00547, SWS_E2E_00548, SWS_E2E_00549, SWS_E2E_00550, SWS_E2E_00551, SWS_E2E_00552, SWS_E2E_00555, SWS_E2E_00556, SWS_E2E_00557, SWS_E2E_00558, SWS_E2E_00559, SWS_E2E_00560, SWS_E2E_00561, SWS_E2E_00562, SWS_E2E_00563, SWS_E2E_00564, SWS_E2E_00565, SWS_E2E_00567, SWS_E2E_00568, SWS_E2E_00569, SWS_E2E_00570, SWS_E2E_00571, SWS_E2E_00572, SWS_E2E_00573, SWS_E2E_00574, SWS_E2E_00575, SWS_E2E_00576, SWS_E2E_00577, SWS_E2E_00578, SWS_E2E_00579, SWS_E2E_00580, SWS_E2E_00581, SWS_E2E_00583, SWS_E2E_00584, SWS_E2E_00585, SWS_E2E_00586, SWS_E2E_00587, SWS_E2E_00588, SWS_E2E_00589, SWS_E2E_00590, SWS_E2E_00591, SWS_E2E_10002, SWS_E2E_10004, SWS_E2E_10005, SWS_E2E_91021, UC_E2E_00053, UC_E2E_00089, UC_E2E_00165, UC_E2E_00192, UC_E2E_00202, UC_E2E_00203, UC_E2E_00204, UC_E2E_00205, UC_E2E_00206, UC_E2E_00207, UC_E2E_00209, UC_E2E_00213, UC_E2E_00230, UC_E2E_00232, UC_E2E_00233, UC_E2E_00235, UC_E2E_00239, UC_E2E_00242, UC_E2E_00248, UC_E2E_00249, UC_E2E_00250, UC_E2E_00251, UC_E2E_00256, UC_E2E_00257, UC_E2E_00258, UC_E2E_00261, UC_E2E_00262, UC_E2E_00263, UC_E2E_00264, UC_E2E_00265, UC_E2E_00266, UC_E2E_00267, UC_E2E_00268, UC_E2E_00270, UC_E2E_00271, UC_E2E_00272, UC_E2E_00273, UC_E2E_00274,</p>
--	--	---

		UC_E2E_00275, UC_E2E_00277, UC_E2E_00278, UC_E2E_00279, UC_E2E_00280, UC_E2E_00288, UC_E2E_00289, UC_E2E_00290, UC_E2E_00292, UC_E2E_00293, UC_E2E_00296, UC_E2E_00297, UC_E2E_00300, UC_E2E_00301, UC_E2E_00313
RS_E2E_08534	E2E protocol shall provide E2E Check status to the application	SWS_E2E_00021, SWS_E2E_00022, SWS_E2E_00047, SWS_E2E_00154, SWS_E2E_00214, SWS_E2E_00336, SWS_E2E_00337, SWS_E2E_00439, SWS_E2E_00440, SWS_E2E_00444, SWS_E2E_00445, SWS_E2E_00542, SWS_E2E_00563, SWS_E2E_00564, SWS_E2E_00568, SWS_E2E_00569, SWS_E2E_00591, SWS_E2E_91008, SWS_E2E_91009, SWS_E2E_91019, SWS_E2E_91022
RS_E2E_08535	-	SWS_E2E_NA_00294
RS_E2E_08539	An E2E protection mechanism for inter-ECU communication of short to large data shall be provided	SWS_E2E_00334, SWS_E2E_00335, SWS_E2E_00336, SWS_E2E_00338, SWS_E2E_00339, SWS_E2E_00349, SWS_E2E_00350, SWS_E2E_00373, SWS_E2E_00377, SWS_E2E_00378, SWS_E2E_00385, SWS_E2E_00393, SWS_E2E_00437, SWS_E2E_00438, SWS_E2E_00439, SWS_E2E_00440, SWS_E2E_00441, SWS_E2E_00443, SWS_E2E_00444, SWS_E2E_00445, SWS_E2E_00446, SWS_E2E_00447, SWS_E2E_00448, SWS_E2E_00449, SWS_E2E_00450, SWS_E2E_00451, SWS_E2E_00452, SWS_E2E_00453, SWS_E2E_00455, SWS_E2E_00456, SWS_E2E_00457, SWS_E2E_00458, SWS_E2E_00459, SWS_E2E_00460, SWS_E2E_00461, SWS_E2E_00542, SWS_E2E_00544, SWS_E2E_00545, SWS_E2E_00546, SWS_E2E_00547, SWS_E2E_00548, SWS_E2E_00549, SWS_E2E_00550, SWS_E2E_00551, SWS_E2E_00552, SWS_E2E_00584, SWS_E2E_00585, SWS_E2E_00586, SWS_E2E_00590, SWS_E2E_00592, SWS_E2E_00593, SWS_E2E_00594, SWS_E2E_00595, SWS_E2E_00596, SWS_E2E_00597, SWS_E2E_00598, SWS_E2E_00599, SWS_E2E_10001, SWS_E2E_10002, SWS_E2E_10004, SWS_E2E_10005, SWS_E2E_91001, SWS_E2E_91002, SWS_E2E_91003, SWS_E2E_91004, SWS_E2E_91005, SWS_E2E_91006, SWS_E2E_91008, SWS_E2E_91010, SWS_E2E_91011, SWS_E2E_91012, SWS_E2E_91013, SWS_E2E_91014, SWS_E2E_91015, SWS_E2E_91017, SWS_E2E_91018, SWS_E2E_91019, SWS_E2E_91020

RS_E2E_08548	E2E protocol shall provide E2E overall state to the application	SWS_E2E_00340, SWS_E2E_00342, SWS_E2E_00343, SWS_E2E_00344, SWS_E2E_00347, SWS_E2E_00351, SWS_E2E_00352, SWS_E2E_00380, SWS_E2E_00381, SWS_E2E_00383, SWS_E2E_00384, SWS_E2E_00453, SWS_E2E_00454, SWS_E2E_00461, SWS_E2E_00462, SWS_E2E_00476, SWS_E2E_00477, SWS_E2E_00553, SWS_E2E_00554, SWS_E2E_00557, SWS_E2E_00558, SWS_E2E_00561, SWS_E2E_00562, SWS_E2E_10003, SWS_E2E_10006, SWS_E2E_10007
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_E2E_00032
SRS_BSW_00004	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	SWS_E2E_00038
SRS_BSW_00005	Modules of the $\hat{\mu}$ C Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_E2E_NA_00294
SRS_BSW_00006	The source code of software modules above the $\hat{\mu}$ C Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_E2E_NA_00294
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_E2E_NA_00294
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_E2E_NA_00294
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_E2E_NA_00294
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_E2E_00037
SRS_BSW_00158	-	SWS_E2E_NA_00294
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_E2E_00037
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_E2E_NA_00294
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides	SWS_E2E_NA_00294

	a standardized interface to higher software layers	
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_E2E_NA_00294
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_E2E_NA_00294
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_E2E_00037
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_E2E_NA_00294
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_E2E_00037
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_E2E_00037
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_E2E_NA_00294
SRS_BSW_00300	All AUTOSAR Basic Software Modules shall be identified by an unambiguous name	SWS_E2E_NA_00294
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_E2E_NA_00294
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_E2E_NA_00294
SRS_BSW_00304	All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types	SWS_E2E_NA_00294
SRS_BSW_00305	Data types naming convention	SWS_E2E_NA_00294
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_E2E_NA_00294
SRS_BSW_00307	Global variables naming convention	SWS_E2E_NA_00294
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global	SWS_E2E_NA_00294

	data in their header files, but in the C file	
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_E2E_NA_00294
SRS_BSW_00310	API naming convention	SWS_E2E_NA_00294
SRS_BSW_00312	Shared code shall be reentrant	SWS_E2E_NA_00294
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_E2E_NA_00294
SRS_BSW_00318	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	SWS_E2E_NA_00294
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_E2E_NA_00294
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_E2E_00047
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_E2E_NA_00294
SRS_BSW_00327	Error values naming convention	SWS_E2E_NA_00294
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_E2E_NA_00294
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_E2E_NA_00294
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_E2E_NA_00294
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_E2E_NA_00294
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_E2E_NA_00294
SRS_BSW_00335	Status values naming convention	SWS_E2E_NA_00294
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_E2E_NA_00294
SRS_BSW_00337	Classification of development errors	SWS_E2E_00047

SRS_BSW_00339	Reporting of production relevant error status	SWS_E2E_00216, SWS_E2E_NA_00294
SRS_BSW_00341	Module documentation shall contain all needed informations	SWS_E2E_NA_00294
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_E2E_NA_00294
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_E2E_NA_00294
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_E2E_00037
SRS_BSW_00345	BSW Modules shall support pre-compile configuration	SWS_E2E_00037
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_E2E_NA_00294
SRS_BSW_00347	A Naming separation of different instances of BSW drivers shall be in place	SWS_E2E_NA_00294
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_E2E_NA_00294
SRS_BSW_00350	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.	SWS_E2E_NA_00294
SRS_BSW_00351	Encapsulation of compiler specific methods to map objects	SWS_E2E_NA_00294
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_E2E_NA_00294
SRS_BSW_00357	For success/failure of an API call a standard return type shall be defined	SWS_E2E_NA_00294
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_E2E_NA_00294
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_E2E_NA_00294

SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_E2E_NA_00294
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_E2E_NA_00294
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_E2E_00049, SWS_E2E_NA_00294
SRS_BSW_00371	The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules	SWS_E2E_NA_00294
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_E2E_NA_00294
SRS_BSW_00374	All Basic Software Modules shall provide a readable module vendor identification	SWS_E2E_NA_00294
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_E2E_NA_00294
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_E2E_NA_00294
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_E2E_NA_00294
SRS_BSW_00379	All software modules shall provide a module identifier in the header file and in the module XML description file.	SWS_E2E_NA_00294
SRS_BSW_00380	Configuration parameters being stored in memory shall be placed into separate c-files	SWS_E2E_NA_00294
SRS_BSW_00381	-	SWS_E2E_NA_00294
SRS_BSW_00383	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	SWS_E2E_NA_00294
SRS_BSW_00384	The Basic Software Module specifications shall specify at least in the description which other modules they require	SWS_E2E_NA_00294
SRS_BSW_00385	List possible error notifications	SWS_E2E_NA_00294
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_E2E_NA_00294

SRS_BSW_00388	Containers shall be used to group configuration parameters that are defined for the same object	SWS_E2E_NA_00294
SRS_BSW_00389	Containers shall have names	SWS_E2E_NA_00294
SRS_BSW_00390	Parameter content shall be unique within the module	SWS_E2E_NA_00294
SRS_BSW_00392	Parameters shall have a type	SWS_E2E_NA_00294
SRS_BSW_00393	Parameters shall have a range	SWS_E2E_NA_00294
SRS_BSW_00394	The Basic Software Module specifications shall specify the scope of the configuration parameters	SWS_E2E_NA_00294
SRS_BSW_00395	The Basic Software Module specifications shall list all configuration parameter dependencies	SWS_E2E_NA_00294
SRS_BSW_00396	The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container	SWS_E2E_NA_00294
SRS_BSW_00397	The configuration parameters in pre-compile time are fixed before compilation starts	SWS_E2E_NA_00294
SRS_BSW_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_E2E_NA_00294
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_E2E_NA_00294
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_E2E_NA_00294
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_E2E_NA_00294
SRS_BSW_00402	Each module shall provide version information	SWS_E2E_NA_00294
SRS_BSW_00403	The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets	SWS_E2E_NA_00294
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_E2E_NA_00294
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_E2E_NA_00294

SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_E2E_NA_00294
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_E2E_NA_00294
SRS_BSW_00408	All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule	SWS_E2E_NA_00294
SRS_BSW_00409	All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration	SWS_E2E_NA_00294
SRS_BSW_00410	Compiler switches shall have defined values	SWS_E2E_NA_00294
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_E2E_NA_00294
SRS_BSW_00412	-	SWS_E2E_NA_00294
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_E2E_NA_00294
SRS_BSW_00414	Init functions shall have a pointer to a configuration structure as single parameter	SWS_E2E_NA_00294
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_E2E_NA_00294
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_E2E_NA_00294
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_E2E_NA_00294
SRS_BSW_00419	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	SWS_E2E_NA_00294
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_E2E_NA_00294

SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_E2E_NA_00294
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_E2E_NA_00294
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_E2E_NA_00294
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_E2E_NA_00294
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_E2E_NA_00294
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_E2E_NA_00294
SRS_BSW_00429	Access to OS is restricted	SWS_E2E_NA_00294
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_E2E_NA_00294
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_E2E_NA_00294
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_E2E_NA_00294
SRS_BSW_00438	Configuration data shall be defined in a structure	SWS_E2E_NA_00294
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_E2E_NA_00294
SRS_BSW_00440	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	SWS_E2E_NA_00294
SRS_BSW_00441	Naming convention for type, macro and function	SWS_E2E_NA_00294
SRS_BSW_00447	Standardizing Include file structure of BSW Modules Implementing Autosar Service	SWS_E2E_NA_00294
SRS_BSW_00448	Module SWS shall not contain requirements from Other Modules	SWS_E2E_NA_00294

SRS_BSW_00449	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	SWS_E2E_NA_00294
SRS_BSW_00450	A Main function of a uninitialized module shall return immediately	SWS_E2E_NA_00294
SRS_BSW_00451	Hardware registers shall be protected if concurrent access to these registers occur	SWS_E2E_NA_00294
SRS_BSW_00452	Classification of runtime errors	SWS_E2E_NA_00294
SRS_BSW_00453	BSW Modules shall be harmonized	SWS_E2E_NA_00294
SRS_BSW_00454	An alternative interface without a parameter of category DATA_REFERENCE shall be available.	SWS_E2E_NA_00294
SRS_BSW_00456	A Header file shall be defined in order to harmonize BSW Modules	SWS_E2E_NA_00294
SRS_BSW_00457	Callback functions of Application software components shall be invoked by the Basis SW	SWS_E2E_NA_00294
SRS_BSW_00458	Classification of production errors	SWS_E2E_NA_00294
SRS_BSW_00459	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	SWS_E2E_NA_00294
SRS_BSW_00460	Reentrancy Levels	SWS_E2E_NA_00294
SRS_BSW_00461	Modules called by generic modules shall satisfy all interfaces requested by the generic module	SWS_E2E_NA_00294
SRS_BSW_00462	All Standardized Autosar Interfaces shall have unique requirement Id / number	SWS_E2E_NA_00294
SRS_BSW_00463	Naming convention of callout prototypes	SWS_E2E_NA_00294
SRS_BSW_00464	File names shall be considered case sensitive regardless of the filesystem in which they are used	SWS_E2E_NA_00294
SRS_BSW_00465	It shall not be allowed to name any two files so that they only differ by the cases of their letters	SWS_E2E_NA_00294
SRS_BSW_00466	Classification of extended production errors	SWS_E2E_NA_00294
SRS_BSW_00467	The init / deinit services shall only be called by BswM or EcuM	SWS_E2E_NA_00294

SRS_BSW_00469	Fault detection and healing of production errors and extended production errors	SWS_E2E_NA_00294
SRS_BSW_00470	Execution frequency of production error detection	SWS_E2E_NA_00294
SRS_BSW_00471	Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors	SWS_E2E_NA_00294
SRS_BSW_00472	Avoid detection of two production errors with the same root cause.	SWS_E2E_NA_00294
SRS_BSW_00473	Classification of transient faults	SWS_E2E_NA_00294
SRS_BSW_00477	The functional interfaces of AUTOSAR BSW modules shall be specified in C90	SWS_E2E_NA_00294
SRS_BSW_00478	Timing limits of main functions	SWS_E2E_NA_00294
SRS_BSW_00479	Interfaces for handling request from external devices	SWS_E2E_NA_00294
SRS_BSW_00480	NullPointer Errors shall follow a naming rule	SWS_E2E_NA_00294
SRS_BSW_00481	Invalid configuration set selection errors shall follow a naming rule	SWS_E2E_NA_00294
SRS_BSW_00482	Get Version Informationfunction shall follow a naming rule	SWS_E2E_NA_00294
SRS_BSW_00483	BSW Modules shall handle buffer alignments internally	SWS_E2E_NA_00294
SRS_LIBS_00001	The functional behavior of each library functions shall not be configurable	SWS_E2E_NA_00294
SRS_LIBS_00002	A library shall be operational before all BSW modules and application SW-Cs	SWS_E2E_NA_00294
SRS_LIBS_00003	A library shall be operational until the shutdown	SWS_E2E_NA_00294
SRS_LIBS_00004	Using libraries shall not pass through a port interface	SWS_E2E_NA_00294
SRS_LIBS_00005	Each library shall provide one header file with its public interface	SWS_E2E_NA_00294
SRS_LIBS_00007	Using a library should be documented	SWS_E2E_NA_00294
SRS_LIBS_00008	For a given function prototype name, the behavior and the parameters shall not evolve once it is a part of an AUTOSAR final release	SWS_E2E_NA_00294
SRS_LIBS_00009	All library functions shall be re-entrant	SWS_E2E_NA_00294

SRS_LIBS_00010	A library shall define its own specific types in the library header file if and only if they are not yet defined by AUTOSAR	SWS_E2E_NA_00294
SRS_LIBS_00011	All function names and type names shall start with "Library short name_"	SWS_E2E_NA_00294
SRS_LIBS_00012	Passing parameters with structure shall be allowed	SWS_E2E_NA_00294
SRS_LIBS_00013	The error cases, resulting in the check at runtime of the value of input parameters, shall be listed in SWS	SWS_E2E_NA_00294
SRS_LIBS_00015	It shall be possible to configure the microcontroller so that the library code is shared between all callers	SWS_E2E_NA_00294
SRS_LIBS_00016	A SW-C may use a non-AUTOSAR library available on the market	SWS_E2E_NA_00294
SRS_LIBS_00017	Usage of macros should be avoided	SWS_E2E_NA_00294
SRS_LIBS_00018	A library function may only call library functions	SWS_E2E_NA_00294
SRS_LIBS_08518	The CRC Library shall provide different calculation methods, optimizing either performance or memory usage	SWS_E2E_NA_00294
SRS_LIBS_08521	All CRC routines shall allow step-by-step-wise calculation of a large data block	SWS_E2E_NA_00294
SRS_LIBS_08525	The CRC library shall support the standard generator polynomials	SWS_E2E_NA_00294
SRS_LIBS_08526	The CRC Library shall support current standards of CRC calculation	SWS_E2E_NA_00294

7 Functional specification

7.1 Error classification

Libraries have no configuration and therefore a tracing of development errors cannot be disabled or enabled. Thus, there is no possibility to classify errors detected by library-internal mechanisms as development or production errors. Moreover, Libraries cannot call BSW modules (e.g. DEM or DET). Therefore, the errors detected by library-internal mechanisms are reported to callers synchronously. Note that both CRC Library and E2E Library are not BSW Modules; Libraries are allowed to call each other.

[SWS_E2E_00049]「The E2E library shall not contain library-internal mechanisms for error detection to be traced as development errors.」(SRS_BSW_00369)

[SWS_E2E_00011]「The E2E Library shall report errors detected by library-internal mechanisms to callers of E2E functions through return value.」(RS_E2E_08528)

[SWS_E2E_00216]「The E2E Library shall not call BSW modules for error reporting (in particular DEM and DET), nor for any other purpose. The E2E Library shall not call RTE.」(SRS_BSW_00339)

7.1.1 Development Errors

The following error flags for errors shall be used by all E2E Library functions:

[SWS_E2E_00047]

<i>Type of error</i>	<i>Related error code</i>	<i>Error value</i>
At least one pointer parameter is a NULL pointer	E2E_E_INPUTERR_NULL	0x13
At least one input parameter is erroneous, e.g. out of range	E2E_E_INPUTERR_WRONG	0x17
An internal library error has occurred (e.g. error detected by program flow monitoring, violated invariant or postcondition)	E2E_E_INTERR	0x19
Function executed in wrong state	E2E_E_WRONGSTATE	0x1A

](SRS_BSW_00337, SRS_BSW_00323, RS_E2E_08534)

The range 0x80..0xFE is foreseen only for extending the AUTOSAR profiles with vendor specific return values.

[UC_E2E_00313] The caller of the E2E functions E2E_PXXProtect() / E2E_PXXCheck() shall handle the errors/status defined in SWS_E2E_00047 according to the column "How do caller of E2E shall handle it". (RS_E2E_08528)

In other words, the E2E library does not define any integration errors for itself. However, the caller of E2E library uses the return values of E2E functions and does the corresponding error handling.

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Transient Faults

There are no transient faults.

7.1.4 Production Errors

There are no production errors.

7.1.5 Extended Production Errors

There are no extended production errors.

8 API specification

This chapter specifies the API of E2E Library.

Members of the configuration structures (e.g. in Figure 8-1) are in alphabetical order. However, for implementation, the sequence of members of this data structure is provided by table specification items (e.g. [SWS_E2E_00018]).

8.1 Imported types

In this chapter, all types and #defines included from the following files are listed:

[SWS_E2E_00017][

<i>Module</i>	<i>Header File</i>	<i>Imported Type</i>
Std	Std_Types.h	Std_MessageResultType
	Std_Types.h	Std_MessageTypeType
	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

](RS_E2E_08528)

8.2 Type definitions

This chapter defines the data types defined by E2E Library that are visible to the callers.

Some attributes shown below define data offset. The offset is defined according to the following rules:

1. The offset is in bits,
2. Within a byte, bits are numbered from 0 upwards, with bit 0 being the least significant bit (regardless of the microcontroller or bus endianness).

Example 1 - Counter with bit offset = 8 on MSB microcontroller:

	MSB							LSB
Data[0]	7	6	5	4	3	2	1	0
	CRC with bit offset 0							
Data[1]	15	14	13	12	11	10	9	8
	User data with bit offset 12				Counter with offset 8			
Data[2]	23	22	21	20	19	18	17	16
	User data with bit offset 20				User data with bit offset 16			

8.2.1 E2E Profile 1 types

Note: Since AUTOSAR 4.1.1, type names were renamed. If an existing application using E2E Library requires compatibility of interfaces to previous release versions, then the header file E2E.h shall contain following type definitions:

```
typedef E2E_P01ProtectStateType E2E_P01SenderStateType;
typedef E2E_P01CheckStateType E2E_P01ReceiverStateType;
typedef E2E_P01CheckStatusType E2E_P01ReceiverStatusType;
```

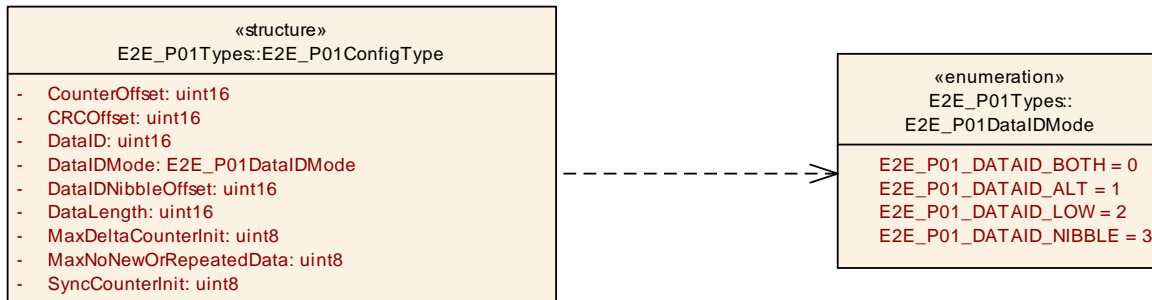


Figure 8-1: E2E Profile 1 configuration

8.2.1.1 E2E_P01ConfigType

[SWS_E2E_00018]

Name	E2E_P01ConfigType	
Kind	Structure	
Elements	CounterOffset	
	Type	uint16
	Comment	Bit offset of Counter in MSB first order. CounterOffset shall be a multiple of 4. In variants 1A, 1B, and 1C, CounterOffset is 8.
	CRCOffset	
	Type	uint16
	Comment	Bit offset of CRC (i.e. since *Data) in MSB first order. The offset shall be a multiple of 8. In variants 1A, 1B, and 1C, CRCOffset is 0.
	DataID	
	Type	uint16
	Comment	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section "Configuration constraints on Data IDs".
	DataIDNibbleOffset	
	Type	uint16
	Comment	Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P01_DATAID_NIBBLE (otherwise it is ignored by E2E Library).

		For DataIDMode different than E2E_P01_DATAID_NIBBLE, DataIDNibbleOffset shall be initialized to 0 (even if it is ignored by E2E Library).
	DataIDMode	
	Type	E2E_P01DataIDMode
	Comment	Inclusion mode of ID in CRC computation (both bytes, alternating, or low byte only of ID included).
	DataLength	
	Type	uint16
	Comment	Length of data, in bits. The value shall be a multiple of 8.
	MaxDeltaCounterInit	
	Type	uint8
	Comment	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	MaxNoNewOrRepeatedData	
	Type	uint8
	Comment	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	SyncCounterInit	
	Type	uint8
	Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 1. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

](RS_E2E_08528)

8.2.1.2 E2E_P01DataIDMode

Note: The values for the enumeration constants are specified on the associated UML diagram.

[SWS_E2E_00200][

Name	E2E_P01DataIDMode
-------------	-------------------

Kind	Enumeration		
Range	E2E_P01_DATAID_BOTH	0	Two bytes are included in the CRC (double ID configuration) This is used in E2E variant 1A.
	E2E_P01_DATAID_ALT	1	One of the two bytes byte is included, alternating high and low byte, depending on parity of the counter (alternating ID configuration). For an even counter, the low byte is included. For an odd counter, the high byte is included. This is used in E2E variant 1B.
	E2E_P01_DATAID_LOW	2	Only the low byte is included, the high byte is never used. This is applicable if the IDs in a particular system are 8 bits.
	E2E_P01_DATAID_NIBBLE	3	The low byte is included in the implicit CRC calculation, the low nibble of the high byte is transmitted along with the data (i.e. it is explicitly included), the high nibble of the high byte is not used. This is applicable for the IDs up to 12 bits. This is used in E2E variant 1C.
Description	The Data ID is two bytes long in E2E Profile 1. There are four inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.		
Available via	E2E.h		

](RS_E2E_08528)

8.2.1.3 E2E_P01ProtectStateType

[SWS_E2E_00020]

Name	E2E_P01ProtectStateType		
Kind	Structure		
Elements	Counter		
	Type	uint8	
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that the first Data will have the counter 0. After the protection by the Counter, the Counter is incremented modulo 0xF. The value 0xF is skipped (after 0xE the next is 0x0), as 0xF value represents the error value. The four high bits are always 0.	
Description	State of the sender for a Data protected with E2E Profile 1.		
Available via	E2E.h		

](RS_E2E_08528)

8.2.1.4 E2E_P01CheckStateType

Note: The values for the enumeration constants are specified on the associated UML diagram. Note that in previous SWS E2E versions, E2E_P01STATUS_OK was equal to 0x10.

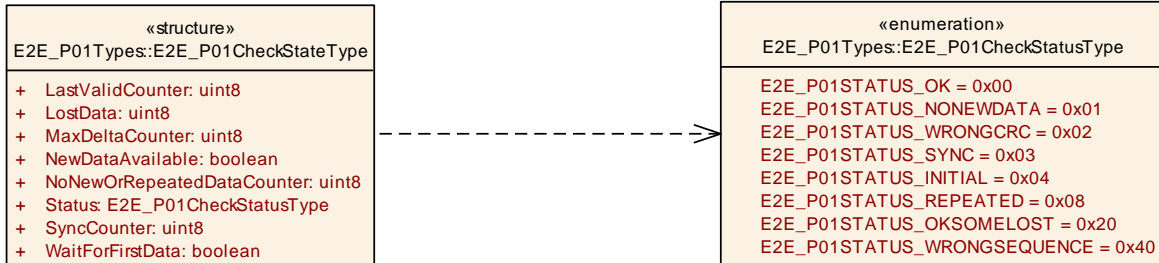


Figure 8-2: E2E Profile 1 check state type

[SWS_E2E_00021]

Name	E2E_P01CheckStateType	
Kind	Structure	
Elements	LastValidCounter	
	Type	uint8
	Comment	Counter value most recently received. If no data has been yet received, then the value is 0x0. After each reception, the counter is updated with the value received.
	MaxDeltaCounter	
	Type	uint8
	Comment	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
	WaitForFirstData	
	Type	boolean
	Comment	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	NewDataAvailable	
	Type	boolean
	Comment	Indicates to E2E Library that a new data is available for Library to be checked. This attribute is set by the E2E Library caller, and not by the E2E Library.
	LostData	
	Type	uint8
Comment	Number of data (messages) lost since reception of last valid one. This attribute is set only if Status equals E2E_P01STATUS_OK or E2E_P01STATUS_OKSOMELOST. For other values of Status, the value of	

	LostData is undefined. E2E_P01CheckStatusType Status Result of the verification of the Data, determined by the Check function.
	Status
Type	E2E_P01CheckStatusType
Comment	Result of the verification of the Data, determined by the Check function.
	SyncCounter
Type	uint8
Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
	NoNewOrRepeatedDataCounter
Type	uint8
Comment	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.
Description	State of the receiver for a Data protected with E2E Profile 1.
Available via	E2E.h

](RS_E2E_08528, RS_E2E_08534)

8.2.1.5 E2E_P01CheckStatusType

[SWS_E2E_00022]

Name	E2E_P01CheckStatusType		
Kind	Enumeration		
Range	E2E_P01STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P01STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P01STATUS_WRONGCRC	0x02	Error: The data has been received according to communication medium, but 1. the CRC is incorrect (applicable for all E2E Profile 1 configurations) or 2. the low nibble of the high byte of Data ID is incorrect (applicable only for E2E Profile 1 with E2E_P01Data IDMode = E2E_P01_DATAID_NIBBLE).

			The two above errors can be a result of corruption, incorrect addressing or masquerade.
	E2E_P01STATUS_SYNC	0x03	NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
	E2E_P01STATUS_INITIAL	0x04	Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.
	E2E_P01STATUS_REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
	E2E_P01STATUS_OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P01STATUS_WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description	Result of the verification of the Data in E2E Profile 1, determined by the Check function.		
Available via	E2E.h		

](RS_E2E_08534)

8.2.2 E2E Profile 2 types

Since AUTOSAR 4.1.1, type names were renamed. If an existing application using E2E Library requires compatibility of interfaces to previous release versions, then the header file E2E.h shall contain following type definitions:

```
typedef E2E_P02ProtectStateType E2E_P02SenderStateType;
typedef E2E_P02CheckStateType E2E_P02ReceiverStateType;
typedef E2E_P02CheckStatusType E2E_P02ReceiverStatusType;
```

8.2.2.1 E2E_P02ConfigType

[SWS_E2E_00152]

Name	E2E_P02ConfigType	
Kind	Structure	
Elements	DataLength	
	Type	uint16
	Comment	Length of Data, in bits. The value shall be a multiple of 8.
	DataIDList	
	Type	Array of uint8
	Size	16
	Comment	An array of appropriately chosen Data IDs for protection against masquerading.
	MaxDeltaCounterInit	
	Type	uint8
	Comment	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	MaxNoNewOrRepeatedData	
	Type	uint8
	Comment	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	SyncCounterInit	
	Type	uint8
Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.	
Offset		
Type	uint16	
Comment	Offset of the E2E header in the Data[] array in bits. It shall be: $0 \leq \text{Offset} \leq \text{DataLength} - (2 \cdot 8)$.	
Description	Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 2. The position of the counter and CRC is not configurable in profile 2.	

Available via	E2E.h
----------------------	-------

](RS_E2E_08528)

8.2.2.2 E2E_P02ProtectStateType

[SWS_E2E_00153]

Name	E2E_P02ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the Data. The initial value is 0. As the counter is incremented before sending, the first Data will have the counter value 1
Description	State of the sender for a Data protected with E2E Profile 2.	
Available via	E2E.h	

](RS_E2E_08528)

8.2.2.3 E2E_P02CheckStateType

Note that in previous SWS E2E versions, E2E_P02STATUS_OK was equal to 0x10.

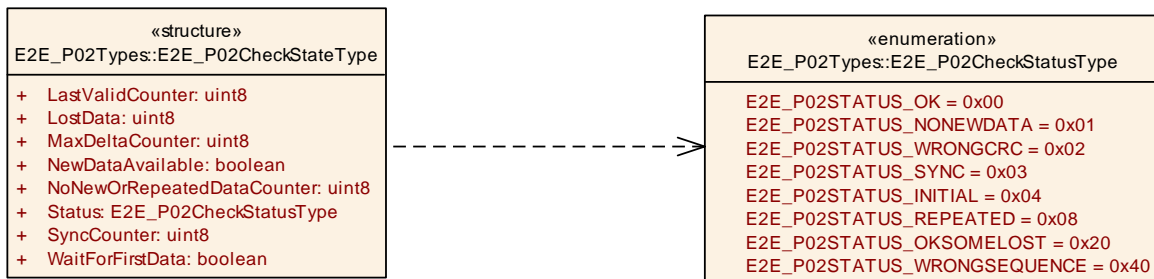


Figure 8-3: E2E Profile 2 check state

[SWS_E2E_00154]

Name	E2E_P02CheckStateType	
Kind	Structure	
Elements	LastValidCounter	
	Type	uint8
	Comment	Counter of last valid received message.

	MaxDeltaCounter
Type	uint8
Comment	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
	WaitForFirstData
Type	boolean
Comment	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	NewDataAvailable
Type	boolean
Comment	Indicates to E2E Library that a new data is available for Library to be checked. This attribute is set by the E2E Library caller, and not by the E2E Library.
	LostData
Type	uint8
Comment	Number of data (messages) lost since reception of last valid one.
	Status
Type	E2E_P02CheckStatusType
Comment	Result of the verification of the Data, determined by the Check function.
	SyncCounter
Type	uint8
Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
	NoNewOrRepeatedDataCounter
Type	uint8
Comment	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.
Description	State of the sender for a Data protected with E2E Profile 2.
Available via	E2E.h

J(RS_E2E_08528, RS_E2E_08534)

8.2.2.4 E2E_P02CheckStatusType

Note: The values for the enumeration constants are specified on the associated UML diagram.

[SWS_E2E_00214]

Name	E2E_P02CheckStatusType		
Kind	Enumeration		
Range	E2E_P02STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P02STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P02STATUS_WRONGCRC	0x02	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P02STATUS_SYNC	0x03	NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
	E2E_P02STATUS_INITIAL	0x04	Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.
	E2E_P02STATUS_REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
	E2E_P02STATUS_OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P02STATUS_WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in

			the sequence have been probably lost since the last correct/initial reception.
Description	Result of the verification of the Data in E2E Profile 2, determined by the Check function.		
Available via	E2E.h		

](RS_E2E_08534)

8.2.3 E2E Profile 4 types

«structure» E2E_P04Types: E2E_P04ConfigType
+ DataID: uint32 + MaxDataLength: uint16 + MaxDeltaCounter: uint16 + MinDataLength: uint16 + Offset: uint16

Figure 8-4: E2E Profile 4 configuration

8.2.3.1 E2E_P04ConfigType

[SWS_E2E_00334][

Name	E2E_P04ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint16
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (12 \cdot 8)$. Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint16
Comment	Minimal length of Data, in bits. E2E checks that Length is \geq MinData Length.	

	MaxDataLength	
	Type	uint16
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength.
	MaxDeltaCounter	
	Type	uint16
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 4. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539)

8.2.3.2 E2E_P04ProtectStateType

[SWS E2E_00335][

Name	E2E_P04ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P04Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 4.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539)

8.2.3.3 E2E_P04CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

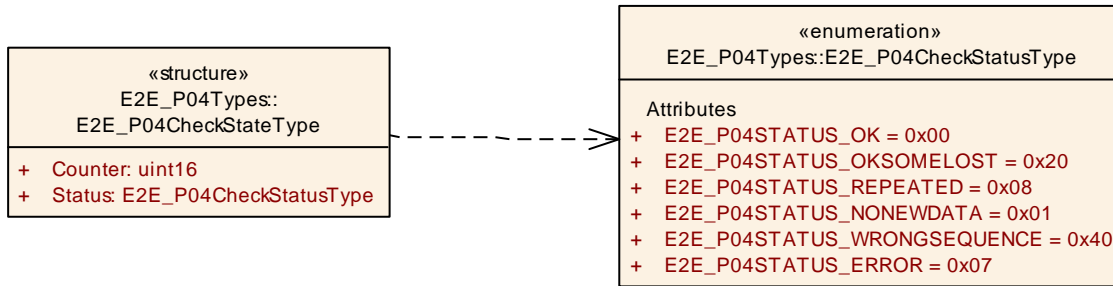


Figure 8-5: E2E Profile 4check state

[SWS_E2E_00336]

Name	E2E_P04CheckStateType		
Kind	Structure		
Elements	Status		
	Type	E2E_P04CheckStatusType	
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.	
	Counter		
	Type	uint16	
Comment	Counter of the data in previous cycle.		
Description	State of the reception on one single Data protected with E2E Profile 4.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08534)

8.2.3.4 E2E_P04CheckStatusType

[SWS_E2E_00337]

Name	E2E_P04CheckStatusType		
Kind	Enumeration		
Range	E2E_P04STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P04STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P04STATUS_REPEATED.

	E2E_P04STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P04STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P04STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P04STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 4.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08534)

Note that the status E2E_P04STATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.4 E2E Profile 4m types

8.2.4.1 E2E_P04mConfigType

<pre> «structure» E2E_P04Types: E2E_P04ConfigType </pre>
<pre> + DataID: uint32 + MaxDataLength: uint16 + MaxDeltaCounter: uint16 + MinDataLength: uint16 + Offset: uint16 </pre>

Figure 8-6: E2E Profile 4m configuration

[SWS_E2E_91021]

Name	E2E_P04mConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
Type	uint16	

	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (12 \cdot 8)$. Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint16
	Comment	Minimal length of Data, in bits. E2E checks that Length is \geq MinDataLength. The value shall be $\leq 4096 \cdot 8$ (4kB) and shall be $\geq 12 \cdot 8$
	MaxDataLength	
	Type	uint16
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is \leq MinDataLength. The value shall be $\leq 4096 \cdot 8$ (4kB) and it shall be \geq MinDataLength
	MaxDeltaCounter	
	Type	uint16
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 4m. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

](RS_E2E_08528)

8.2.4.2 E2E_P04mProtectStateType

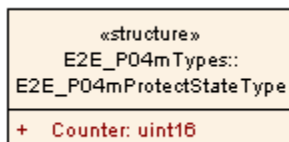


Figure 8-7: E2E Profile 4m Protect state type

[SWS_E2E_91020]

Name	E2E_P04mProtectStateType
Kind	Structure

Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P04m Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 4m.	
Available via	E2E.h	

](RS_E2E_08539)

8.2.4.3 E2E_P04mCheckStateType

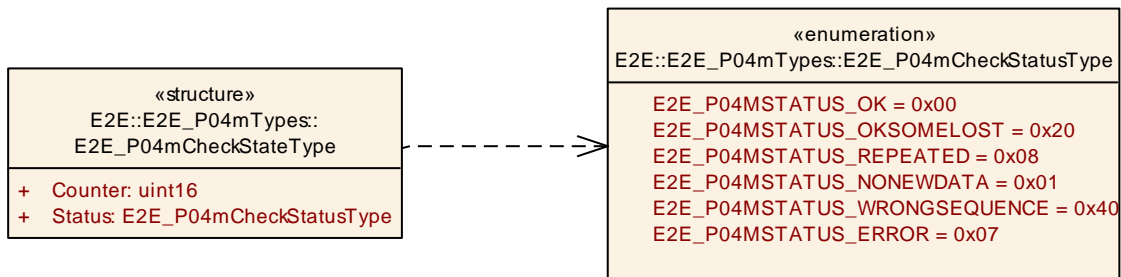


Figure 8-8: E2E Profile 4m check state

[SWS E2E_91019]

Name	E2E_P04mCheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P04mCheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint16
Comment	Counter of the data in previous cycle.	
Description	State of the reception on one single Data protected with E2E Profile 4m.	

Available via	E2E.h
----------------------	-------

](RS_E2E_08539, RS_E2E_08534)

8.2.4.4 E2E_P04mCheckStatusType

[SWS_E2E_91022][

Name	E2E_P04mCheckStatusType		
Kind	Enumeration		
Range	E2E_P04MSTATUS_OK	0x00	--
	E2E_P04MSTATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P04STATUS_REPEATED.
	E2E_P04MSTATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P04MSTATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P04MSTATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P04MSTATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 4m.		
Available via	E2E.h		

](RS_E2E_08534)

Note that the status E2E_P04MSTATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.5 E2E Profile 5 types

8.2.5.1 E2E_P05ConfigType

<pre>«structure» E2E_P05Types: E2E_P05ConfigType</pre>
<pre>+ DataID: uint16 + DataLength: uint16 + MaxDeltaCounter: uint8 + Offset: uint16</pre>

Figure 8-9: E2E Profile 5 configuration

[SWS E2E_00437]

Name	E2E_P05ConfigType	
Kind	Structure	
Elements	Offset	
	Type	uint16
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{DataLength} - (3 \cdot 8)$. Example: If Offset equals 8, then the low byte of the E2E Crc (16 bit) is written to Byte 1, the high Byte is written to Byte 2.
	DataLength	
	Type	uint16
	Comment	Length of data, in bits
	DataID	
	Type	uint16
	Comment	A system-unique identifier of the Data
	MaxDeltaCounter	
	Type	uint8
Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.	
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 5. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539)

8.2.5.2 E2E_P05ProtectStateType

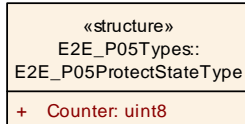


Figure 8-10: E2E Profile 5 Protect state type

[SWS_E2E_00438][

Name	E2E_P05ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P05Protect() is called, it increments the counter up to 0xFF.
Description	State of the sender for a Data protected with E2E Profile 5.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539)

8.2.5.3 E2E_P05CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

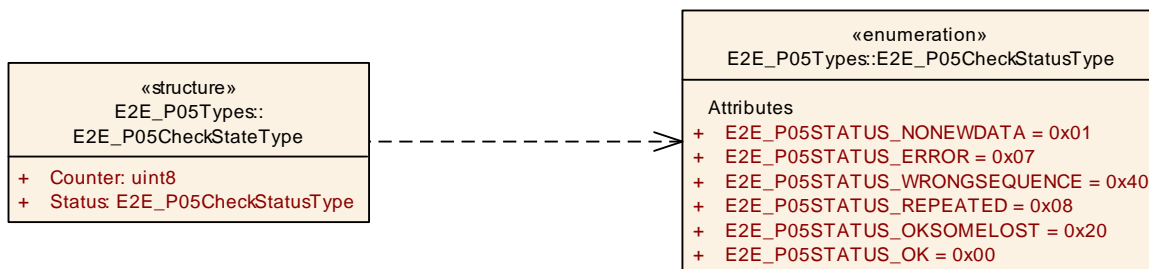


Figure 8-11: E2E Profile 5 Check state type

[SWS_E2E_00439][

Name	E2E_P05CheckStateType
Kind	Structure
Elements	Status

	Type	E2E_P05CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint8
	Comment	Counter of the data in previous cycle.
Description	Description: State of the reception on one single Data protected with E2E Profile 5.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08534)

8.2.5.4 E2E_P05CheckStatusType

[SWS_E2E_00440]

Name	E2E_P05CheckStatusType		
Kind	Enumeration		
Range	E2E_P05STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P05STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P05STATUS_REPEATED.
	E2E_P05STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length).
	E2E_P05STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P05STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P05STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 5.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08534)

8.2.6 E2E Profile 6 types

8.2.6.1 E2E_P06ConfigType

«structure» E2E_P06Types: E2E_P06ConfigType
+ DataID: uint16
+ MaxDataLength: uint16
+ MaxDeltaCounter: uint8
+ MinDataLength: uint16
+ Offset: uint16

Figure 8-12: E2E Profile 6 configuration

[SWS_E2E_00441]

Name	E2E_P06ConfigType	
Kind	Structure	
Elements	Offset	
	Type	uint16
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (5 \cdot 8)$. Example: If Offset equals 8, then the high byte of the E2E Crc (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint16
	Comment	Minimal length of Data, in bits. E2E checks that Length is \geq MinData Length.
	MaxDataLength	
	Type	uint16
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is \leq Max DataLength.
	DataID	
	Type	uint16
	Comment	A system-unique identifier of the Data
	MaxDeltaCounter	
	Type	uint8
Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.	

Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 6. For each transmitted Data, there is an instance of this typedef.
Available via	E2E.h

](RS_E2E_08528, RS_E2E_08539)

8.2.6.2 E2E_P06ProtectStateType

«structure» E2E_P06Types: E2E_P06ProtectStateType
+ Counter: uint8

Figure 8-13: E2E Profile 6 Protect state type

[SWS_E2E_00443][

Name	E2E_P06ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P06Protect() is called, it increments the counter up to 0xFF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 6.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539)

8.2.6.3 E2E_P06CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

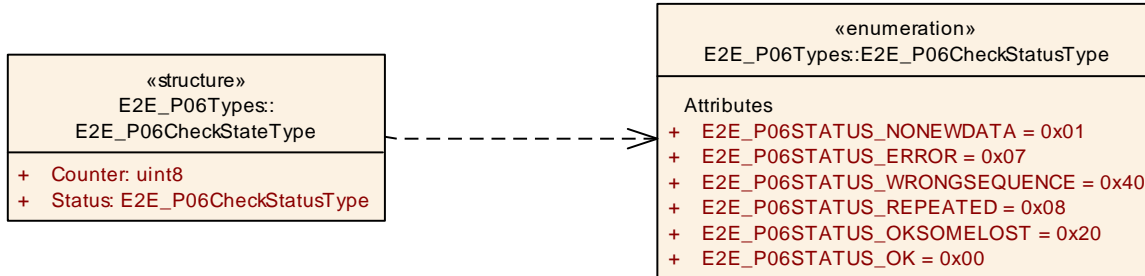


Figure 8-14: E2E Profile 6 Check state type

[SWS E2E_00444]

Name	E2E_P06CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P06CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint8
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 6.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08534)

8.2.6.4 E2E_P06CheckStatusType

[SWS_E2E_00445][

Name	E2E_P06CheckStatusType		
Kind	Enumeration		
Range	E2E_P06STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P06STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P06STATUS_REPEATED.
	E2E_P06STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length).
	E2E_P06STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P06STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P06STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 6.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08534)

8.2.7 E2E Profile 7 types

8.2.7.1 E2E_P07ConfigType

«structure» E2E_P07Types: E2E_P07ConfigType
+ DataID: uint32 + MaxDataLength: uint32 + MaxDeltaCounter: uint32 + MinDataLength: uint32 + Offset: uint32

Figure 8-15: E2E Profile 7 configuration

[SWS_E2E_00544]

Name	E2E_P07ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (20 \cdot 8)$. Example: If Offset equals 8, then the first byte of the E2E Length (32 bit) is written to byte 1, the next byte is written to byte 2 and so on.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that Length is \geq MinData Length.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is \leq MinDataLength. The value shall be \geq MinDataLength
	MaxDeltaCounter	
	Type	uint32
Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.	

Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 7. For each transmitted Data, there is an instance of this typedef.
Available via	E2E.h

](RS_E2E_08528, RS_E2E_08539)

8.2.7.2 E2E_P07ProtectStateType

«structure» E2E_P07Types: E2E_P07ProtectStateType + Counter: uint32
--

Figure 8-16: E2E Profile 7 Protect state type

[SWS E2E_00545][

Name	E2E_P07ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P07Protect() is called, it increments the counter up to 0xFF'FF'FF'FF.
Description	State of the sender for a Data protected with E2E Profile 7.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539)

8.2.7.3 E2E_P07CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

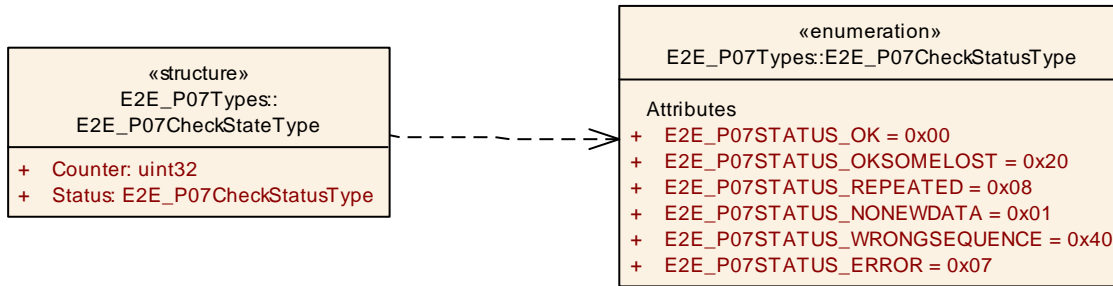


Figure 8-17: E2E Profile 7 Check state type

[SWS_E2E_00542][

Name	E2E_P07CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P07CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint32
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 7.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08534)

8.2.7.4 E2E_P07CheckStatusType

[SWS_E2E_00591][

Name	E2E_P07CheckStatusType		
Kind	Enumeration		
Range	E2E_P07STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P07STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P07STATUS_REPEATED.

	E2E_P07STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P07STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P07STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P07STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 7.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08534)

8.2.8 E2E Profile 7m types

8.2.8.1 E2E_P07mConfigType

<pre> «structure» E2E_P07mTypes:: E2E_P07mConfigType + DataID: uint32 + MaxDataLength: uint32 + MaxDeltaCounter: uint32 + MinDataLength: uint32 + Offset: uint32 </pre>

Figure 8-18: E2E Profile 7m configuration

[SWS_E2E_91010]

Name	E2E_P07mConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (12 \cdot 8)$. Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that Length is $\geq \text{MinDataLength}$. The value shall be $\geq 20 \cdot 8$ and $\leq \text{MaxDataLength}$.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is $\leq \text{MinDataLength}$. The value shall be $\leq 4096 \cdot 8$ (4kB) and it shall be $\geq \text{MinDataLength}$
	MaxDeltaCounter	
	Type	uint32
Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1	

		and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 7m. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

](RS_E2E_08539)

8.2.8.2 E2E_P07mProtectStateType

<pre> «structure» E2E_P07mTypes:: E2E_P07mProtectStateType + Counter: uint32 </pre>

Figure 8-19: E2E Profile 7m Protect state type

[SWS_E2E_91011][

Name	E2E_P07mProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P07m Protect() is called, it increments the counter up to 0xFF'FF'FF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 7m.	
Available via	E2E.h	

](RS_E2E_08539)

8.2.8.3 E2E_P07mCheckStateType

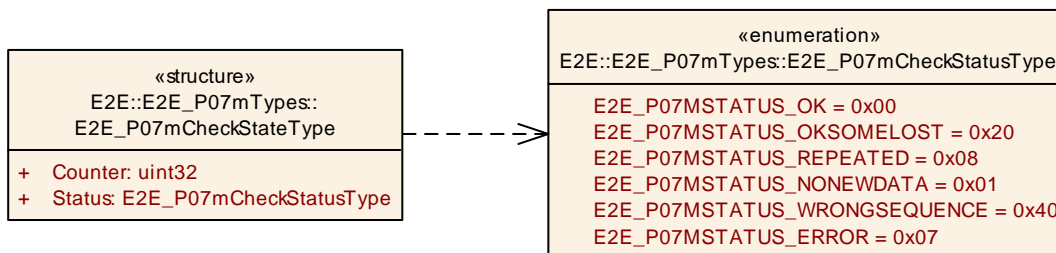


Figure 8-20: E2E Profile 7m Check state type

[SWS_E2E_91008]

Name	E2E_P07mCheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P07mCheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint32
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 7m.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08534)

8.2.8.4 E2E_P07mCheckStatusType

[SWS_E2E_91009]

Name	E2E_P07mCheckStatusType		
Kind	Enumeration		
Range	E2E_P07MSTATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P07MSTATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE,

			COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P07STATUS_REPEATED.
	E2E_P07MSTATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P07MSTATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P07MSTATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P07MSTATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 7m.		
Available via	E2E.h		

](RS_E2E_08534)

8.2.9 E2E Profile 8 types

8.2.9.1 E2E_P08ConfigType

«structure» E2E::E2E_P08Types: E2E_P08ConfigType
+ DataID: uint32 + MaxDataLength: uint32 + MaxDeltaCounter: uint32 + MinDataLength: uint32 + Offset: uint32

Figure 8-21: E2E Profile 08 configuration

[SWS E2E_91033]

Name	E2E_P08ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32

	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (16 \cdot 8)$. Example: If Offset equals 8, then the high byte of the E2E Length (32 bit) is written to byte 1, the next byte is written to byte 2 and so on.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that Length is \geq MinData Length. The value shall be $\geq 16 \cdot 8$ and $\leq \text{MaxDataLength}$.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is \leq Max DataLength. The value shall be $\geq \text{MinDataLength}$
	MaxDeltaCounter	
	Type	uint32
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 08. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

l)

8.2.9.2 E2E_P08ProtectStateType

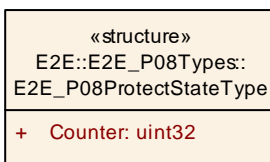


Figure 8-22: E2E Profile 08 Protect state type

[SWS_E2E_91042]

Name	E2E_P08ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P08Protect() is called, it increments the counter up to 0xFF'FF'FF'FF.
Description	State of the sender for a Data protected with E2E Profile 08.	
Available via	E2E.h	

]()

8.2.9.3 E2E_P08CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

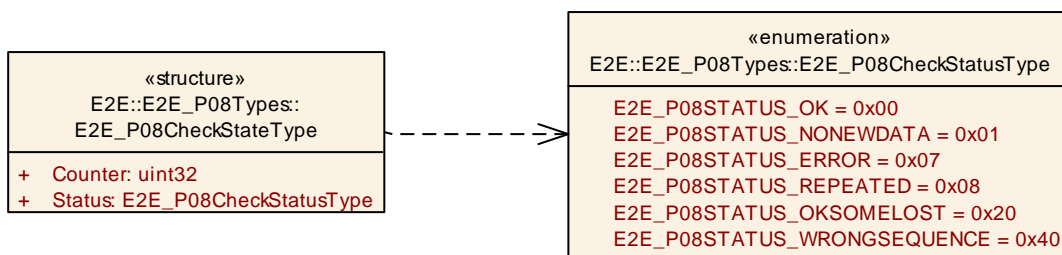


Figure 8-23: E2E Profile 08 Check state type

[SWS E2E 91034]

Name	E2E_P08CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P08CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint32
	Comment	Counter of the data in previous cycle.

Description	State of the reception on one single Data protected with E2E Profile 08
Available via	E2E.h

]()

8.2.9.4 E2E_P08CheckStatusType

[SWS_E2E_91035]

Name	E2E_P08CheckStatusType		
Kind	Enumeration		
Range	E2E_P08STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P08STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P08STATUS_REPEATED.
	E2E_P08STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P08STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P08STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P08STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 08.		
Available via	E2E.h		

]()

Note that the status E2E_P08STATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.10 E2E Profile 11 types

8.2.10.1 E2E_P11ConfigType

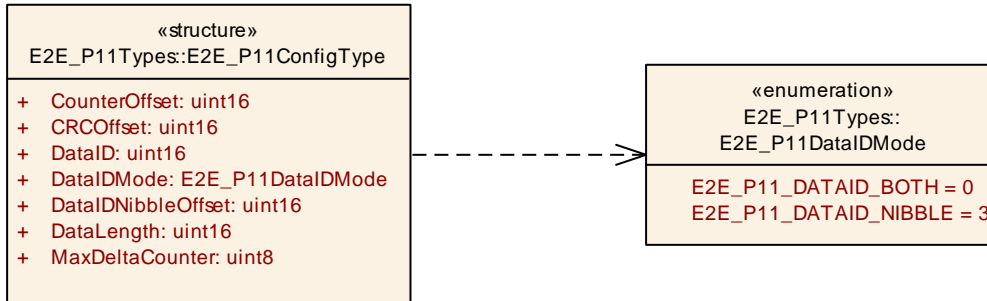


Figure 8-24: E2E Profile 11 configuration

[SWS_E2E_00565]

Name	E2E_P11ConfigType	
Kind	Structure	
Elements	DataLength	
	Type	uint16
	Comment	Length of data, in bits. The value shall be a multiple of 8.
	DataID	
	Type	uint16
	Comment	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section "Configuration constraints on Data IDs".
	MaxDeltaCounter	
	Type	uint8
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
	DataIDMode	
	Type	E2E_P11DataIDMode
	Comment	--
	CRCOffset	
	Type	uint16
Comment	Bit offset of CRC (i.e. since *Data) in MSB first order. In variants 1A and 1B, CRCOffset is 0. The offset shall be a multiple of 8.	

	CounterOffset	
	Type	uint16
	Comment	Bit offset of Counter in MSB first order. In variants 1A and 1B, Counter Offset is 8. The offset shall be a multiple of 4.
	DataIDNibbleOffset	
	Type	uint16
	Comment	Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P11_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than E2E_P11_DATAID_NIBBLE, Data IDNibbleOffset shall be initialized to 0 (even if it is ignored by E2E Library).
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 11. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

](RS_E2E_08528)

8.2.10.2 E2E_P11ProtectStateType

«structure» E2E_P11Types: E2E_P11ProtectStateType
+ Counter: uint8

Figure 8-25: E2E Profile 11 Protect state type

[SWS_E2E_00567]

Name	E2E_P11ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P11Protect() is called, it increments the counter up to 0x0E.
Description	State of the sender for a Data protected with E2E Profile 11.	
Available via	E2E.h	

](RS_E2E_08528)

8.2.10.3 E2E_P11CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

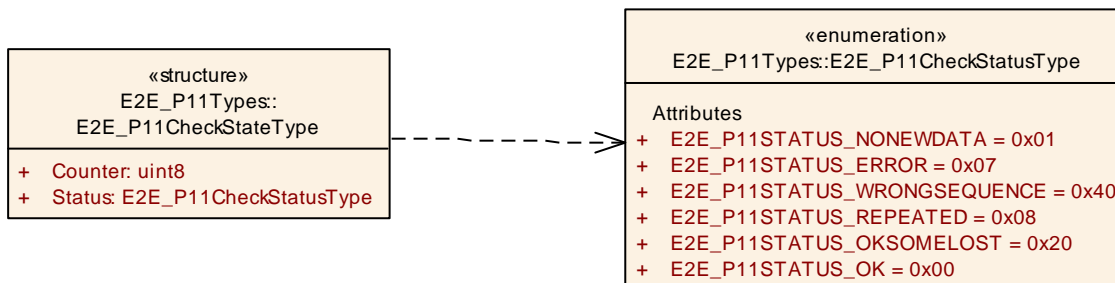


Figure 8-26: E2E Profile 11 Check state type

[SWS E2E_00563]

Name	E2E_P11CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P11CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint8
	Comment	Counter of the data in previous cycle.
Description	Description: State of the reception on one single Data protected with E2E Profile 11.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08534)

8.2.10.4 E2E_P11CheckStatusType

[SWS_E2E_00564][

Name	E2E_P11CheckStatusType		
Kind	Enumeration		
Range	E2E_P11STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P11STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P11STATUS_REPEATED.
	E2E_P11STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length).
	E2E_P11STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P11STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P11STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 11.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08534)

8.2.11 E2E Profile 22 types

8.2.11.1 E2E_P22ConfigType

«structure» E2E_P22Types: E2E_P22ConfigType
+ DataIDList: uint8[16] + DataLength: uint16 + MaxDeltaCounter: uint8 + Offset: uint16

Figure 8-27: E2E Profile 22 configuration

[SWS_E2E_00571]

Name	E2E_P22ConfigType	
Kind	Structure	
Elements	DataLength	
	Type	uint16
	Comment	Length of Data, in bits. The value shall be a multiple of 8.
	DataIDList	
	Type	Array of uint8
	Size	16
	Comment	An array of appropriately chosen Data IDs for protection against masquerading.
	MaxDeltaCounter	
	Type	uint8
	Comment	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	Offset	
	Type	uint16
Comment	Offset of the E2E header in the Data[] array in bits. It shall be: $0 \leq \text{Offset} \leq \text{MaxDataLength} - (2 \cdot 8)$.	
Description	Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 22. The position of the counter and CRC is not configurable in profile 22.	

Available via	E2E.h
----------------------	-------

](RS_E2E_08528)

8.2.11.2 E2E_P22ProtectStateType

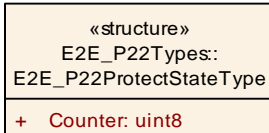


Figure 8-28: E2E Profile 22 Protect state type

[SWS_E2E_00570][

Name	E2E_P22ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the Data. The initial value is 0, which means that the first Data will have the counter 0. After the protection by the counter, the counter is incremented modulo 16.
Description	State of the sender for a Data protected with E2E Profile 22.	
Available via	E2E.h	

](RS_E2E_08528)

8.2.11.3 E2E_P22CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).

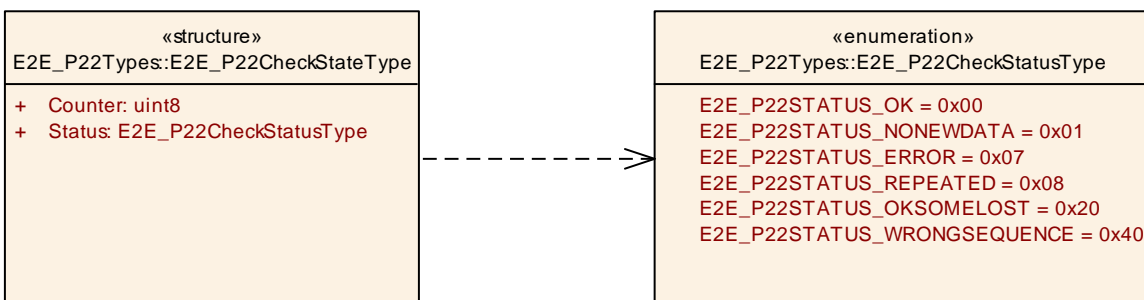


Figure 8-29: E2E Profile 22 Check state type

[SWS_E2E_00568][

Name	E2E_P22CheckStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter of last valid received message.
	Status	
	Type	E2E_P22CheckStatusType
	Comment	Result of the verification of the Data, determined by the Check function.
Description	State of the sender for a Data protected with E2E Profile 22.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08534)

8.2.11.4 E2E_P22CheckStatusType

[SWS_E2E_00569]

Name	E2E_P22CheckStatusType		
Kind	Enumeration		
Range	E2E_P22STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P22STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P22STATUS_ERROR	0x07	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P22STATUS_REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
	E2E_P22STATUS_OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P22STATUS_WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description	Result of the verification of the Data in E2E Profile 22, determined by the Check function.		
Available via	E2E.h		

](RS_E2E_08528, RS_E2E_08534)

8.2.12 E2E Profile 44 types

8.2.12.1 E2E_P44ConfigType

<pre>«structure» E2E_P44Types:: E2E_P44ConfigType</pre>
<pre>+ DataID: uint32 + MaxDataLength: uint32 + MaxDeltaCounter: uint16 + MinDataLength: uint32 + Offset: uint32</pre>

Figure 8-30: E2E Profile 44 configuration

[SWS E2E 91023]

Name	E2E_P44ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and $0 \leq \text{Offset} \leq \text{MaxDataLength} - (12 \cdot 8)$. Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that Length is \geq MinData Length
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is \leq Max DataLength.
	MaxDeltaCounter	
	Type	uint16
Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4	

Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 44. For each transmitted Data, there is an instance of this typedef.
Available via	E2E.h

]()

8.2.12.2 E2E_P44ProtectStateType

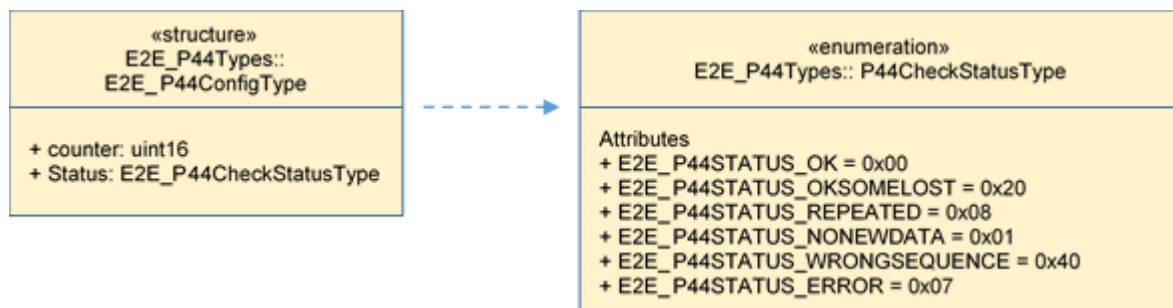
[SWS_E2E_91024]

Name	E2E_P44ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P44Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 44.	
Available via	E2E.h	

]()

8.2.12.3 E2E_P44CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).



[SWS_E2E_91025]

Name	E2E_P44CheckStateType
-------------	-----------------------

Kind	Structure	
Elements	Status	
	Type	E2E_P44CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint16
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 44.	
Available via	E2E.h	

l()

8.2.12.4 E2E_P44CheckStatusType

[SWS E2E 91026]

Name	E2E_P44CheckStatusType		
Kind	Enumeration		
Range	E2E_P44STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P44STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P44STATUS_REPEATED.
	E2E_P44STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P44STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P44STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P44STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta

Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 44.
Available via	E2E.h

J()

Note that the status E2E_P44STATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.13 E2E state machine types

8.2.13.1 E2E_PCheckStatusType

[SWS_E2E_00347][

Name	E2E_PCheckStatusType		
Kind	Enumeration		
Range	E2E_P_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check).
	E2E_P_REPEATED	0x01	Data has a repeated counter.
	E2E_P_WRONGSEQUENCE	0x02	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
	E2E_P_ERROR	0x03	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong Data ID) or the return of the check function was not OK.
	E2E_P_NOTAVAILABLE	0x04	No value has been received yet (e.g. during initialization). This is used as the initialization value for the buffer, it is not returned by any E2E profile.
	E2E_P_NONEWDATA	0x05	No new data is available.
	reserved	0x07, 0x0F	reserved for runtime errors (shall not be used for any status in future).
Description	Profile-independent status of the reception on one single Data in one cycle.		
Available via	E2E.h		

](RS_E2E_08548)

8.2.13.2 E2E_SMConfigType

[SWS_E2E_00342][

Name	E2E_SMConfigType		
Kind	Structure		
Elements	WindowSizeValid		
	Type	uint8	
	Comment	Size of the monitoring window for the state machine during state VALID.	
	MinOkStateInit		
	Type	uint8	

Comment	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_INIT) required to change to state E2E_SM_VALID.
MaxErrorStateInit	
Type	uint8
Comment	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INIT).
MinOkStateValid	
Type	uint8
Comment	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_VALID) required to keep in state E2E_SM_VALID.
MaxErrorStateValid	
Type	uint8
Comment	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_VALID).
MinOkStateInvalid	
Type	uint8
Comment	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_INVALID) required to change to state E2E_SM_VALID.
MaxErrorStateInvalid	
Type	uint8
Comment	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INVALID).
WindowSizeInit	
Type	uint8
Comment	Size of the monitoring windows for the state machine during state INIT.
WindowSizeInvalid	
Type	uint8
Comment	Size of the monitoring window for the state machine during state INVALID.
ClearToInvalid	
Type	boolean
Comment	Clear monitoring window data on transition to state INVALID.

Description	Configuration of a communication channel for exchanging Data.
Available via	E2E.h

](RS_E2E_08548)

8.2.13.3 E2E_SMCheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).



Figure 8-31: E2E SM check state

[SWS_E2E_00343][

Name	E2E_SMCheckStateType	
Kind	Structure	
Elements	ProfileStatusWindow	
	Type	uint8*
	Comment	Pointer to an array, in which the ProfileStatus-es of the last E2E-checks are stored. The array size shall be WindowSize
	WindowTopIndex	
	Type	uint8
	Comment	index in the array, at which the next ProfileStatus is to be written.
	OkCount	
	Type	uint8
	Comment	Count of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks.
	ErrorCount	
Type	uint8	
Comment	Count of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks.	

	SMState	
	Type	E2E_SMStateType
	Comment	The current state in the state machine. The value is not explicitly used in the pseudocode of the state machine, because it is expressed in UML as UML states.
Description	State of the protection of a communication channel.	
Available via	E2E.h	

](RS_E2E_08548)

8.2.13.4 E2E_SMStateType

[SWS_E2E_00344][

Name	E2E_SMStateType		
Kind	Enumeration		
Range	E2E_SM_VALID	0x00	Communication functioning properly according to E2E, data can be used.
	E2E_SM_DEINIT	0x01	State before E2E_SMCheckInit() is invoked, data cannot be used.
	E2E_SM_NODATA	0x02	No data from the sender is available since the initialization, data cannot be used.
	E2E_SM_INIT	0x03	There has been some data received since startup, but it is not yet possible use it, data cannot be used.
	E2E_SM_INVALID	0x04	Communication not functioning properly, data cannot be used.
	reserved	0x07, 0x0F	reserved for runtime errors (shall not be used for any state in future)
Description	Status of the communication channel exchanging the data. If the status is OK, then the data may be used.		
Available via	E2E.h		

](RS_E2E_08548)

8.3 Routine definitions

This chapter defines the routines provided by E2E Library. The provided routines can be implemented as:

1. Functions
2. Inline functions
3. Macros

8.3.1 E2E Profile 1 routines

8.3.1.1 E2E_P01Protect

[SWS_E2E_00166][

Service Name	E2E_P01Protect	
Syntax	<pre>Std_ReturnType E2E_P01Protect (const E2E_P01ConfigType* ConfigPtr, E2E_P01ProtectStateType* StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.1.2 E2E_P01ProtectInit

[SWS_E2E_00385]

Service Name	E2E_P01ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P01ProtectInit (E2E_P01ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x1b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00386] In case State is NULL, E2E_P01ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08528)

8.3.1.3 E2E_P01Forward

[SWS_E2E_00588]{DRAFT} [

Service Name	E2E_P01Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P01Forward (const E2E_P01ConfigType* ConfigPtr, E2E_PCheckStatusType Checkstatus, E2E_P01ProtectStateType* StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x38	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Checkstatus	E2E Status of the received message

Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08527, RS_E2E_08528)

8.3.1.4 E2E_P01Check

[SWS_E2E_00158][

Service Name	E2E_P01Check	
Syntax	<pre>Std_ReturnType E2E_P01Check (const E2E_P01ConfigType* Config, E2E_P01CheckStateType* State, const uint8* Data)</pre>	
Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Config	Pointer to static configuration.
	Data	Pointer to received data.
Parameters (inout)	State	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.	

Available via	E2E.h
----------------------	-------

](RS_E2E_08528, RS_E2E_08527)

8.3.1.5 E2E_P01CheckInit

[SWS_E2E_00390]

Service Name	E2E_P01CheckInit	
Syntax	Std_ReturnType E2E_P01CheckInit (E2E_P01CheckStateType* StatePtr)	
Service ID [hex]	0x1c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00389] In case State is NULL, E2E_P01CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. LastValidCounter = 0
2. MaxDeltaCounter = 0
3. WaitForFirstData = TRUE
4. NewDataAvailable = TRUE
5. LostData = 0
6. Status = E2E_P01STATUS_NONEWDATA
7. NoNewOrRepeatedDataCounter = 0
8. SyncCounter = 0.](RS_E2E_08528)

The LastValidCounter is ignored in the first cycle(s) because WaitForFirstData is set to TRUE, therefore the value does not need to be set to 0xE.

8.3.1.6 E2E_P01MapStatusToSM

[SWS_E2E_00382]

Service Name	E2E_P01MapStatusToSM	
Syntax	<pre>E2E_PCheckStatusType E2E_P01MapStatusToSM (Std_ReturnType CheckReturn, E2E_P01CheckStatusType Status, boolean profileBehavior)</pre>	
Service ID [hex]	0x1d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P01Check function
	Status	Status determined by E2E_P01Check function
	profileBehavior	FALSE: check has the legacy behavior, before R4.2 TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheck-StatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

⌋(RS_E2E_08528, RS_E2E_08527)

This represents the R4.2 behavior:

[SWS_E2E_00383] If CheckReturn == E2E_E_OK and ProfileBehavior == TRUE, then the function E2E_P01MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P01STATUS_OK E2E_P01STATUS_OKSOMELOST E2E_P01STATUS_SYNC	E2E_P_OK
E2E_P01STATUS_WRONGCRC	E2E_P_ERROR
E2E_P01STATUS_REPEATED	E2E_P_REPEATED
E2E_P01STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P01STATUS_WRONGSEQUENCE E2E_P01STATUS_INITIAL	E2E_P_WRONGSEQUENCE

⌋(RS_E2E_08528, RS_E2E_08548)

This represents the pre-R4.2 behavior:

[SWS_E2E_00476]

If CheckReturn == E2E_E_OK and ProfileBehavior == FALSE, then the function E2E_P01MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P01STATUS_OK E2E_P01STATUS_OKSOMELOST E2E_P01STATUS_INITIAL	E2E_P_OK
E2E_P01STATUS_WRONGCRC	E2E_P_ERROR
E2E_P01STATUS_REPEATED	E2E_P_REPEATED
E2E_P01STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P01STATUS_WRONGSEQUENCE E2E_P01STATUS_SYNC	E2E_P_WRONGSEQUENCE

](RS_E2E_08528, RS_E2E_08548)

[SWS_E2E_00384] If CheckReturn != E2E_E_OK, then the function E2E_P01MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).](RS_E2E_08528, RS_E2E_08548)

8.3.2 E2E Profile 2 routines

8.3.2.1 E2E_P02Protect

[SWS_E2E_00160]

Service Name	E2E_P02Protect	
Syntax	<pre>Std_ReturnType E2E_P02Protect (const E2E_P02ConfigType* ConfigPtr, E2E_P02ProtectStateType* StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to the data to be protected.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR

		E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.2.2 E2E_P02ProtectInit

[SWS_E2E_00387][

Service Name	E2E_P02ProtectInit	
Syntax	Std_ReturnType E2E_P02ProtectInit (E2E_P02ProtectStateType* StatePtr)	
Service ID [hex]	0x1e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00388] In case State is NULL, E2E_P02ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08528)

8.3.2.3 E2E_P02Forward

[SWS_E2E_00583]{DRAFT} [

Service Name	E2E_P02Forward (draft)	
Syntax	Std_ReturnType E2E_P02Forward (const E2E_P02ConfigType* ConfigPtr, E2E_PCheckStatusType Checkstatus,	

	<pre>E2E_P02ProtectStateType* StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x32	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08527, RS_E2E_08528)

8.3.2.4 E2E_P02Check

[SWS_E2E_00161][

Service Name	E2E_P02Check	
Syntax	<pre>Std_ReturnType E2E_P02Check (const E2E_P02ConfigType* ConfigPtr, E2E_P02CheckStateType* StatePtr, const uint8* DataPtr)</pre>	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	--

Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.2.5 E2E_P02CheckInit

[SWS_E2E_00391][

Service Name	E2E_P02CheckInit	
Syntax	Std_ReturnType E2E_P02CheckInit (E2E_P02CheckStateType* StatePtr)	
Service ID [hex]	0x1f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00392] In case State is NULL, E2E_P02CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. LastValidCounter = 0
2. MaxDeltaCounter = 0
3. WaitForFirstData = TRUE
4. NewDataAvailable = TRUE

5. LostData = 0
6. Status = E2E_P02STATUS_NONEWDATA
7. NoNewOrRepeatedDataCounter = 0
8. SyncCounter = 0. (RS_E2E_08528)

The LastValidCounter is ignored in the first cycle(s) because WaitForFirstData is set to TRUE, therefore the value does not need to be set to 0xF.

8.3.2.6 E2E_P02MapStatusToSM

[SWS_E2E_00379]

Service Name	E2E_P02MapStatusToSM	
Syntax	<pre>E2E_PCheckStatusType E2E_P02MapStatusToSM (Std_ReturnType CheckReturn, E2E_P02CheckStatusType Status, boolean profileBehavior)</pre>	
Service ID [hex]	0x20	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P02Check function
	Status	Status determined by E2E_P02Check function
	profileBehavior	FALSE: check has the legacy behavior, before R4.2 TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheck-StatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

This represents the R4.2 behavior:

[SWS_E2E_00380] If CheckReturn == E2E_E_OK and ProfileBehavior == 1, then the function E2E_P02MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P02STATUS_OK E2E_P02STATUS_OKSOMELOST E2E_P02STATUS_SYNC	E2E_P_OK
E2E_P02STATUS_WRONGCRC	E2E_P_ERROR
E2E_P02STATUS_REPEATED	E2E_P_REPEATED
E2E_P02STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P02STATUS_WRONGSEQUENCE E2E_P02STATUS_INITIAL	E2E_P_WRONGSEQUENCE

⌋(RS_E2E_08528, RS_E2E_08548)

This represents the pre-R4.2 behavior:

[SWS_E2E_00477]⌈

If CheckReturn == E2E_E_OK and ProfileBehavior == 0, then the function E2E_P02MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P02STATUS_OK E2E_P02STATUS_OKSOMELOST E2E_P02STATUS_INITIAL	E2E_P_OK
E2E_P02STATUS_WRONGCRC	E2E_P_ERROR
E2E_P02STATUS_REPEATED	E2E_P_REPEATED
E2E_P02STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P02STATUS_WRONGSEQUENCE E2E_P02STATUS_SYNC	E2E_P_WRONGSEQUENCE

⌋(RS_E2E_08528, RS_E2E_08548)

[SWS_E2E_00381]⌈ If CheckReturn != E2E_E_OK, then the function E2E_P02MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋(RS_E2E_08528, RS_E2E_08548)

8.3.3 E2E Profile 4 routines

8.3.3.1 E2E_P04Protect

[SWS_E2E_00338]

Service Name	E2E_P04Protect	
Syntax	<pre>Std_ReturnType E2E_P04Protect (const E2E_P04ConfigType* ConfigPtr, E2E_P04ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x21	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 4. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.3.2 E2E_P04ProtectInit

[SWS_E2E_00373]

Service Name	E2E_P04ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P04ProtectInit (E2E_P04ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x22	

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00377] In case State is NULL, E2E_P04ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08539)

8.3.3.3 E2E_P04Forward

[SWS_E2E_00584]{DRAFT} [

Service Name	E2E_P04Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P04Forward (const E2E_P04ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P04ProtectStateType StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x33	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.

Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527, RS_E2E_08539)

8.3.3.4 E2E_P04Check

[SWS_E2E_00339][

Service Name	E2E_P04Check	
Syntax	<pre>Std_ReturnType E2E_P04Check (const E2E_P04ConfigType* ConfigPtr, E2E_P04CheckStateType* StatePtr, const uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x23	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.
Available via	E2E.h

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.3.5 E2E_P04CheckInit

[SWS_E2E_00350]

Service Name	E2E_P04CheckInit	
Syntax	Std_ReturnType E2E_P04CheckInit (E2E_P04CheckStateType* StatePtr)	
Service ID [hex]	0x24	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00378] In case State is NULL, E2E_P04CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF.
2. Status to E2E_P04STATUS_ERROR.](RS_E2E_08528, RS_E2E_08539)

8.3.3.6 E2E_P04MapStatusToSM

[SWS_E2E_00349]

Service Name	E2E_P04MapStatusToSM
---------------------	----------------------

Syntax	E2E_PCheckStatusType E2E_P04MapStatusToSM (Std_ReturnType CheckReturn, E2E_P04CheckStatusType Status)	
Service ID [hex]	0x25	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P04Check function
	Status	Status determined by E2E_P04Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus- Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00351] If CheckReturn = E2E_E_OK, then the function
E2E_P04MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P04STATUS_OK or E2E_P04STATUS_OKSOMELOST	E2E_P_OK
E2E_P04STATUS_ERROR	E2E_P_ERROR
E2E_P04STATUS_REPEATED	E2E_P_REPEATED
E2E_P04STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P04STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

](RS_E2E_08548)

[SWS_E2E_00352] If CheckReturn != E2E_E_OK, then the function
E2E_P04MapStatusToSM() shall return E2E_P_ERROR (regardless of value of
Status).](RS_E2E_08548)

8.3.4 E2E Profile 4m routines

8.3.4.1 E2E_P04mProtect

[SWS_E2E_91005][

Service Name	E2E_P04mProtect	
Syntax	<pre>Std_ReturnType E2E_P04mProtect (const E2E_P04mConfigType* ConfigPtr, E2E_P04mProtectStateType* StatePtr, uint32 SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

8.3.4.2 E2E_P04mProtectInit

[SWS_E2E_91006]

Service Name	E2E_P04mProtectInit
---------------------	---------------------

Syntax	Std_ReturnType E2E_P04mProtectInit (E2E_P04mProtectStateType* StatePtr)	
Service ID [hex]	0x47	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00592] In case State is NULL, E2E_P04mProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08539)

8.3.4.3 E2E_P04mForward

[SWS_E2E_91007]{DRAFT} [

Service Name	E2E_P04mForward (draft)	
Syntax	Std_ReturnType E2E_P04mForward (const E2E_P04mConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P04mProtectStateType* StatePtr, uint32 SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, uint8* DataPtr)	
Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
	ConfigPtr	Pointer to static configuration.

Parameters (in)	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08527, RS_E2E_08539)

8.3.4.4 E2E_P04mSourceCheck

[SWS_E2E_91002][

Service Name	E2E_P04mSourceCheck	
Syntax	<pre>Std_ReturnType E2E_P04mSourceCheck (const E2E_P04mConfigType* ConfigPtr, E2E_P04mCheckStateType* StatePtr, uint32 SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, const uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x43	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.

	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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).	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

8.3.4.5 E2E_P04mSinkCheck

[SWS_E2E_91003]

Service Name	E2E_P04mSinkCheck	
Syntax	<pre>Std_ReturnType E2E_P04mSinkCheck (const E2E_P04mConfigType* ConfigPtr, E2E_P04mCheckStateType StatePtr, uint32* SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, const uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x44	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	MessageType	Type of the message (request/response)

	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	SourceID	A system-unique identifier of the Data Source.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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).	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

8.3.4.6 E2E_P04mCheckInit

[SWS_E2E_91001][

Service Name	E2E_P04mCheckInit	
Syntax	Std_ReturnType E2E_P04mCheckInit (E2E_P04mCheckStateType* StatePtr)	
Service ID [hex]	0x42	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00593] In case State is NULL, E2E_P04mCheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

- 1- Counter to 0xFF'FF.
- 2- Status to E2E_P04MSTATUS_ERROR.](RS_E2E_08539)

8.3.4.7 E2E_P04mMapStatusToSM

[SWS_E2E_91004]

Service Name	E2E_P04mMapStatusToSM	
Syntax	E2E_PCheckStatusType E2E_P04mMapStatusToSM (Std_ReturnType CheckReturn, E2E_P04mCheckStatusType Status)	
Service ID [hex]	0x45	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P04mSinkCheck/E2E_P04mSourceCheck function
	Status	Status determined by E2E_P04mSinkCheck/E2E_P04mSourceCheck function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus-Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00594] If CheckReturn = E2E_E_OK, then the function E2E_P04mMapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P04MSTATUS_OK or E2E_P04MSTATUS_OKSOMELOST	E2E_P_OK
E2E_P04MSTATUS_ERROR	E2E_P_ERROR
E2E_P04MSTATUS_REPEATED	E2E_P_REPEATED
E2E_P04MSTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P04MSTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

](RS_E2E_08539)

[SWS_E2E_00595] If CheckReturn != E2E_E_OK, then the function E2E_P04mMapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).](RS_E2E_08539)

8.3.5 E2E Profile 5 routines

8.3.5.1 E2E_P05Protect

[SWS_E2E_00446]

Service Name	E2E_P05Protect	
Syntax	<pre>Std_ReturnType E2E_P05Protect (const E2E_P05ConfigType* ConfigPtr, E2E_P05ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x26	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 5. This includes checksum calculation, handling of counter.
Available via	E2E.h

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.5.2 E2E_P05ProtectInit

[SWS_E2E_00447][

Service Name	E2E_P05ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P05ProtectInit (E2E_P05ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x27	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00448] In case State is NULL, E2E_P05ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08539)

8.3.5.3 E2E_P05Forward

[SWS_E2E_00585]{DRAFT} [

Service Name	E2E_P05Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P05Forward (const E2E_P05ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P05ProtectStateType* StatePtr, </pre>	

	<pre>uint8* DataPtr)</pre>	
Service ID [hex]	0x34	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527, RS_E2E_08539)

8.3.5.4 E2E_P05Check

[SWS_E2E_00449][

Service Name	E2E_P05Check
Syntax	<pre>Std_ReturnType E2E_P05Check (const E2E_P05ConfigType* ConfigPtr, E2E_P05CheckStateType* StatePtr, const uint8* DataPtr, uint16 Length)</pre>
Service ID [hex]	0x28
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.5.5 E2E_P05CheckInit

[SWS_E2E_00450]

Service Name	E2E_P05CheckInit	
Syntax	Std_ReturnType E2E_P05CheckInit (E2E_P05CheckStateType* StatePtr)	
Service ID [hex]	0x29	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00451] In case State is NULL, E2E_P05CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF
2. Status to E2E_P05STATUS_ERROR. (RS_E2E_08528, RS_E2E_08539)

8.3.5.6 E2E_P05MapStatusToSM

[SWS_E2E_00452]

Service Name	E2E_P05MapStatusToSM	
Syntax	E2E_PCheckStatusType E2E_P05MapStatusToSM (Std_ReturnType CheckReturn, E2E_P05CheckStatusType Status)	
Service ID [hex]	0x2a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P05Check function
	Status	Status determined by E2E_P05Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus-Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

(RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00453] If CheckReturn = E2E_E_OK, then the function E2E_P05MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P05STATUS_OK or E2E_P05STATUS_OKSOMELOST	E2E_P_OK
E2E_P05STATUS_ERROR	E2E_P_ERROR
E2E_P05STATUS_REPEATED	E2E_P_REPEATED
E2E_P05STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P05STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

(RS_E2E_08539, RS_E2E_08548)

[SWS_E2E_00454] If CheckReturn != E2E_E_OK, then the function E2E_P05MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status). (RS_E2E_08548)

8.3.6 E2E Profile 6 routines

8.3.6.1 E2E_P06Protect

[SWS_E2E_00393]

Service Name	E2E_P06Protect	
Syntax	<pre>Std_ReturnType E2E_P06Protect (const E2E_P06ConfigType* ConfigPtr, E2E_P06ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x2b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 6. This includes checksum calculation, handling of counter.	
Available via	E2E.h	

(RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.6.2 E2E_P06ProtectInit

[SWS_E2E_00455]

Service Name	E2E_P06ProtectInit
---------------------	--------------------

Syntax	Std_ReturnType E2E_P06ProtectInit (E2E_P06ProtectStateType* StatePtr)	
Service ID [hex]	0x2c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00456] In case State is NULL, E2E_P06ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0. (RS_E2E_08528, RS_E2E_08539)

8.3.6.3 E2E_P06Forward

[SWS_E2E_00586]{DRAFT} [

Service Name	E2E_P06Forward (draft)	
Syntax	Std_ReturnType E2E_P06Forward (const E2E_P06ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P06ProtectStateType* StatePtr, uint8* DataPtr)	
Service ID [hex]	0x35	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message

Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527, RS_E2E_08539)

8.3.6.4 E2E_P06Check

[SWS_E2E_00457][

Service Name	E2E_P06Check	
Syntax	Std_ReturnType E2E_P06Check (const E2E_P06ConfigType* ConfigPtr, E2E_P06CheckStateType* StatePtr, const uint8* DataPtr, uint16 Length)	
Service ID [hex]	0x2d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR

		E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.6.5 E2E_P06CheckInit

[SWS_E2E_00458][

Service Name	E2E_P06CheckInit	
Syntax	Std_ReturnType E2E_P06CheckInit (E2E_P06CheckStateType* StatePtr)	
Service ID [hex]	0x2e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00459] In case State is NULL, E2E_P06CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF
2. Status to E2E_P06STATUS_ERROR.](RS_E2E_08528, RS_E2E_08539)

8.3.6.6 E2E_P06MapStatusToSM

[SWS_E2E_00460][

Service Name	E2E_P06MapStatusToSM
---------------------	----------------------

Syntax	E2E_PCheckStatusType E2E_P06MapStatusToSM (Std_ReturnType CheckReturn, E2E_P06CheckStatusType Status)	
Service ID [hex]	0x2f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P06Check function
	Status	Status determined by E2E_P06Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus- Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00461] If CheckReturn = E2E_E_OK, then the function
E2E_P06MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P06STATUS_OK or E2E_P06STATUS_OKSOMELOST	E2E_P_OK
E2E_P06STATUS_ERROR	E2E_P_ERROR
E2E_P06STATUS_REPEATED	E2E_P_REPEATED
E2E_P06STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P06STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

](RS_E2E_08539, RS_E2E_08548)

[SWS_E2E_00462] If CheckReturn != E2E_E_OK, then the function
E2E_P06MapStatusToSM() shall return E2E_P_ERROR (regardless of value of
Status).](RS_E2E_08548)

8.3.7 E2E Profile 7 routines

8.3.7.1 E2E_P07Protect

[SWS_E2E_00546][

Service Name	E2E_P07Protect	
Syntax	<pre>Std_ReturnType E2E_P07Protect (const E2E_P07ConfigType* ConfigPtr, E2E_P07ProtectStateType* StatePtr, uint8* DataPtr, uint32 Length)</pre>	
Service ID [hex]	0x21	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 7. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.7.2 E2E_P07ProtectInit

[SWS_E2E_00547]

Service Name	E2E_P07ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P07ProtectInit (E2E_P07ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x22	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	

Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00551] In case State is NULL, E2E_P07ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08528, RS_E2E_08539)

8.3.7.3 E2E_P07Forward

[SWS_E2E_00590]{DRAFT} [

Service Name	E2E_P07Forward (draft)	
Syntax	Std_ReturnType E2E_P07Forward (E2E_P07ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P07ProtectStateType* StatePtr, uint8* DataPtr)	
Service ID [hex]	0x39	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR

		E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527, RS_E2E_08539)

8.3.7.4 E2E_P07Check

[SWS_E2E_00548][

Service Name	E2E_P07Check	
Syntax	<pre>Std_ReturnType E2E_P07Check (const E2E_P07ConfigType* ConfigPtr, E2E_P07CheckStateType* StatePtr, const uint8* DataPtr, uint32 Length)</pre>	
Service ID [hex]	0x23	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.7.5 E2E_P07CheckInit

[SWS_E2E_00549]

Service Name	E2E_P07CheckInit	
Syntax	Std_ReturnType E2E_P07CheckInit (E2E_P07CheckStateType* StatePtr)	
Service ID [hex]	0x24	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00552] In case State is NULL, E2E_P07CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF'FF'FF
2. Status to E2E_P07STATUS_ERROR.](RS_E2E_08528, RS_E2E_08539)

8.3.7.6 E2E_P07MapStatusToSM

[SWS_E2E_00550]

Service Name	E2E_P07MapStatusToSM	
Syntax	E2E_PCheckStatusType E2E_P07MapStatusToSM (E2E_PCheckStatusType return, E2E_P07CheckStatusType Status)	
Service ID [hex]	0x25	
Sync/Async	Synchronous	

Reentrancy	Reentrant	
Parameters (in)	return	Profile-independent status of the reception on one single Data in one cycle.
	Status	Status determined by E2E_P07Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus-Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

|(RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00553] If CheckReturn = E2E_E_OK, then the function E2E_P07MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P07STATUS_OK or E2E_P07STATUS_OKSOMELOST	E2E_P_OK
E2E_P07STATUS_ERROR	E2E_P_ERROR
E2E_P07STATUS_REPEATED	E2E_P_REPEATED
E2E_P07STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P07STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

|(RS_E2E_08548)

[SWS_E2E_00554] If CheckReturn != E2E_E_OK, then the function E2E_P07MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status). |(RS_E2E_08548)

8.3.8 E2E Profile 7m routines

8.3.8.1 E2E_P07mProtect

[SWS_E2E_91014]

Service Name	E2E_P07mProtect	
Syntax	<pre>Std_ReturnType E2E_P07mProtect (const E2E_P07mConfigType* ConfigPtr, E2E_P07mProtectStateType* StatePtr, uint32 SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x4b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

8.3.8.2 E2E_P07mProtectInit

[SWS_E2E_91015]

Service Name	E2E_P07mProtectInit	
Syntax	<pre>Std_ReturnType E2E_P07mProtectInit (E2E_P07mProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x4c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00596] In case State is NULL, E2E_P07mProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.](RS_E2E_08539)

8.3.8.3 E2E_P07mForward

[SWS_E2E_91016]{DRAFT} [

Service Name	E2E_P07mForward (draft)	
Syntax	<pre>Std_ReturnType E2E_P07mForward (const E2E_P07mConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P07mProtectStateType* StatePtr, uint32 SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, uint8* DataPtr)</pre>	
Service ID [hex]	0x4d	
Sync/Async	Synchronous	

Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08527, RS_E2E_08539)

8.3.8.4 E2E_P07mSourceCheck

[SWS_E2E_91018][

Service Name	E2E_P07mSourceCheck
Syntax	<pre>Std_ReturnType E2E_P07mSourceCheck (const E2E_P07mConfigType* ConfigPtr, E2E_P07mCheckStateType* StatePtr, uint32 SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, const uint8* DataPtr, uint16 Length)</pre>
Service ID [hex]	0x4f
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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).	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

8.3.8.5 E2E_P07mSinkCheck

[SWS_E2E_91017][

Service Name	E2E_P07mSinkCheck
Syntax	Std_ReturnType E2E_P07mSinkCheck (const E2E_P07mConfigType* ConfigPtr, E2E_P07mCheckStateType* StatePtr, uint32* SourceID, Std_MessageTypeType MessageType, Std_MessageResultType MessageResult, const uint8* DataPtr, uint16 Length)
Service ID [hex]	0x4e
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	ConfigPtr	Pointer to static configuration.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	SourceID	A system-unique identifier of the Data Source.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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).	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

8.3.8.6 E2E_P07mCheckInit

[SWS_E2E_91012][

Service Name	E2E_P07mCheckInit	
Syntax	Std_ReturnType E2E_P07mCheckInit (E2E_P07mCheckStateType* StatePtr)	
Service ID [hex]	0x49	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK

Description	Initializes the check state
Available via	E2E.h

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00597] In case State is NULL, E2E_P07mCheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

- 3- Counter to 0xFF'FF.
- 4- Status to E2E_P07MSTATUS_ERROR.](RS_E2E_08539)

8.3.8.7 E2E_P07mMapStatusToSM

[SWS_E2E_91013]

Service Name	E2E_P07mMapStatusToSM	
Syntax	<pre>E2E_PCheckStatusType E2E_P07mMapStatusToSM (Std_ReturnType CheckReturn, E2E_P07mCheckStatusType Status)</pre>	
Service ID [hex]	0x4a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P07mSinkCheck/E2E_P07mSourceCheck function
	Status	Status determined by E2E_P07mSinkCheck/E2E_P07mSourceCheck function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus-Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

](RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00598] If CheckReturn = E2E_E_OK, then the function E2E_P07mMapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P07MSTATUS_OK or E2E_P07MSTATUS_OKSOMELOST	E2E_P_OK
E2E_P07MSTATUS_ERROR	E2E_P_ERROR
E2E_P07MSTATUS_REPEATED	E2E_P_REPEATED
E2E_P07MSTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P07MSTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

](RS_E2E_08539)

[SWS_E2E_00599] If CheckReturn != E2E_E_OK, then the function E2E_P07mMapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).](RS_E2E_08539)

8.3.9 E2E Profile 8 routines

8.3.9.1 E2E_P08Protect

[SWS_E2E_91036]

Service Name	E2E_P08Protect	
Syntax	<pre>Std_ReturnType E2E_P08Protect (const E2E_P08ConfigType* ConfigPtr, E2E_P08ProtectStateType* StatePtr, uint8* DataPtr, uint32 Length)</pre>	
Service ID [hex]	0x57	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	DataPtr	Pointer to Data to be transmitted.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 08. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

l()

8.3.9.2 E2E_P08ProtectInit

[SWS_E2E_91037]

Service Name	E2E_P08ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P08ProtectInit (E2E_P08ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x58	

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Initializes the protection state.	
Available via	E2E.h	

]()

[SWS_E2E_10004] In case State is NULL, E2E_P08ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0. (RS_E2E_08528, RS_E2E_08539)

8.3.9.3 E2E_P08Forward

[SWS_E2E_91038]{DRAFT} [

Service Name	E2E_P08Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P08Forward (const E2E_P08ConfigType* ConfigPtr, uint32 Length, E2E_PCheckStatusType Checkstatus, E2E_P08ProtectStateType StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x59	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.

Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

]()

8.3.9.4 E2E_P08Check

[SWS_E2E_91039][

Service Name	E2E_P08Check	
Syntax	<pre>Std_ReturnType E2E_P08Check (const E2E_P08ConfigType* ConfigPtr, E2E_P08CheckStateType* StatePtr, const uint8* DataPtr, uint32 Length)</pre>	
Service ID [hex]	0x5a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.

Description	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.
Available via	E2E.h

]()

8.3.9.5 E2E_P08CheckInit

[SWS_E2E_91040][

Service Name	E2E_P08CheckInit	
Syntax	<pre>Std_ReturnType E2E_P08CheckInit (E2E_P08CheckStateType* StatePtr)</pre>	
Service ID [hex]	0x5b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state	
Available via	E2E.h	

]()

[SWS_E2E_10005] In case State is NULL, E2E_P08CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF'FF'FF
2. Status to E2E_P08STATUS_ERROR. (RS_E2E_08528, RS_E2E_08539)

8.3.9.6 E2E_P08MapStatusToSM

[SWS_E2E_91041][

Service Name	E2E_P08MapStatusToSM	
Syntax	<pre>E2E_PCheckStatusType E2E_P08MapStatusToSM (Std_ReturnType return,</pre>	

	E2E_P08CheckStatusType Status)	
Service ID [hex]	0x5c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	return	Return value of the E2E_P08Check function
	Status	Status determined by E2E_P08Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus-Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

l()

[SWS_E2E_10006] If CheckReturn = E2E_E_OK, then the function E2E_P08MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P08STATUS_OK or E2E_P08STATUS_OKSOMELOST	E2E_P_OK
E2E_P08STATUS_ERROR	E2E_P_ERROR
E2E_P08STATUS_REPEATED	E2E_P_REPEATED
E2E_P08STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P08STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

l(RS_E2E_08548)

[SWS_E2E_10007] If CheckReturn != E2E_E_OK, then the function E2E_P08MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status). l(RS_E2E_08548)

8.3.10 E2E Profile 11 routines

8.3.10.1 E2E_P11Protect

[SWS_E2E_00575]

Service Name	E2E_P11Protect	
Syntax	<pre>Std_ReturnType E2E_P11Protect (const E2E_P11ConfigType* ConfigPtr, E2E_P11ProtectStateType StatePtr, uint8 DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x3b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL, E2E_E_OK, E2E_E_INPUTERR_WRONG, For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 11. This includes checksum calculation, handling of counter.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.10.2 E2E_P11ProtectInit

[SWS_E2E_00576]

Service Name	E2E_P11ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P11ProtectInit (E2E_P11ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x3c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	

Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_Return-Type	E2E_E_INPUTERR_NULL - null pointer passed, E2E_E_OK.
Description	Initializes the protection state.	
Available via	E2E.h	

] (RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00555] In case State is NULL, E2E_P11ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0. (RS_E2E_08528)

8.3.10.3 E2E_P11Forward

[SWS_E2E_00587]{DRAFT} [

Service Name	E2E_P11Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P11Forward (const E2E_P11ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P11ProtectStateType* StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x36	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG

		E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08527, RS_E2E_08528)

8.3.10.4 E2E_P11Check

[SWS_E2E_00572][

Service Name	E2E_P11Check	
Syntax	<pre>Std_ReturnType E2E_P11Check (const E2E_P11ConfigType* ConfigPtr, E2E_P11CheckStateType StatePtr, const uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x38	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL, E2E_E_OK, E2E_E_INPUTERR_WRONG, For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.10.5 E2E_P11CheckInit

[SWS_E2E_00573]

Service Name	E2E_P11CheckInit	
Syntax	Std_ReturnType E2E_P11CheckInit (E2E_P11CheckStateType* StatePtr)	
Service ID [hex]	0x39	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_Return- Type	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK.
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00556] In case State is NULL, E2E_P11CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xE
2. Status to E2E_P11STATUS_ERROR.](RS_E2E_08528)

8.3.10.6 E2E_P11MapStatusToSM

[SWS_E2E_00574]

Service Name	E2E_P11MapStatusToSM	
Syntax	E2E_PCheckStatusType E2E_P11MapStatusToSM (Std_ReturnType CheckReturn, E2E_P11CheckStatusType Status)	
Service ID [hex]	0x3a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
	CheckReturn	Return value of the E2E_P11Check function

Parameters (in)	Status	Status determined by E2E_P11Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus- Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

_(RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00557] If CheckReturn = E2E_E_OK, then the function E2E_P11MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P11STATUS_OK or E2E_P11STATUS_OKSOMELOST	E2E_P_OK
E2E_P11STATUS_ERROR	E2E_P_ERROR
E2E_P11STATUS_REPEATED	E2E_P_REPEATED
E2E_P11STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P11STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

_(RS_E2E_08528, RS_E2E_08548)

[SWS_E2E_00558] If CheckReturn != E2E_E_OK, then the function E2E_P11MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status)._(RS_E2E_08528, RS_E2E_08548)

8.3.11 E2E Profile 22 routines

8.3.11.1 E2E_P22Protect

[SWS_E2E_00580]

Service Name	E2E_P22Protect	
Syntax	<pre>Std_ReturnType E2E_P22Protect (const E2E_P22ConfigType* ConfigPtr, E2E_P22ProtectStateType StatePtr, uint8 DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x40	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL, E2E_E_OK, E2E_E_INPUTERR_WRONG, For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 22. This includes checksum calculation, handling of counter.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.11.2 E2E_P22ProtectInit

[SWS_E2E_00581]

Service Name	E2E_P22ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P22ProtectInit (E2E_P22ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x41	
Sync/Async	Synchronous	
Reentrancy	Reentrant	

Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_Return-Type	E2E_E_INPUTERR_NULL - null pointer passed, E2E_E_OK.
Description	Initializes the protection state.	
Available via	E2E.h	

] (RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00559] In case State is NULL, E2E_P22ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.] (RS_E2E_08528)

8.3.11.3 E2E_P22Forward

[SWS_E2E_00589]{DRAFT} [

Service Name	E2E_P22Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P22Forward (E2E_P22ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType Checkstatus, E2E_P22ProtectStateType* StatePtr, uint8 DataPtr)</pre>	
Service ID [hex]	0x37	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG

		E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

](RS_E2E_08527, RS_E2E_08528)

8.3.11.4 E2E_P22Check

[SWS_E2E_00577][

Service Name	E2E_P22Check	
Syntax	<pre>Std_ReturnType E2E_P22Check (const E2E_P22ConfigType* ConfigPtr, E2E_P22CheckStateType StatePtr, const uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x3d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL, E2E_E_OK, E2E_E_INPUTERR_WRONG, For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

8.3.11.5 E2E_P22CheckInit

[SWS_E2E_00578]

Service Name	E2E_P22CheckInit	
Syntax	Std_ReturnType E2E_P22CheckInit (E2E_P22CheckStateType* StatePtr)	
Service ID [hex]	0x3e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_Return- Type	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK.
Description	Initializes the check state	
Available via	E2E.h	

](RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00560] In case State is NULL, E2E_P22CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

3. Counter to 0xF
4. Status to E2E_P22STATUS_ERROR.](RS_E2E_08528)

8.3.11.6 E2E_P22MapStatusToSM

[SWS_E2E_00579]

Service Name	E2E_P22MapStatusToSM	
Syntax	E2E_PCheckStatusType E2E_P22MapStatusToSM (Std_ReturnType CheckReturn, E2E_P22CheckStatusType Status)	
Service ID [hex]	0x3f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
	CheckReturn	Return value of the E2E_P22Check function

Parameters (in)	Status	Status determined by E2E_P22Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatus- Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

_(RS_E2E_08528, RS_E2E_08527)

[SWS_E2E_00561] If CheckReturn = E2E_E_OK, then the function E2E_P22MapStatusToSM shall return the values depending on the value of Status:

Status	Return value
E2E_P22STATUS_OK or E2E_P22STATUS_OKSOMELOST	E2E_P_OK
E2E_P22STATUS_ERROR	E2E_P_ERROR
E2E_P22STATUS_REPEATED	E2E_P_REPEATED
E2E_P22STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P22STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

_(RS_E2E_08528, RS_E2E_08548)

[SWS_E2E_00562] If CheckReturn != E2E_E_OK, then the function E2E_P22MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status)._(RS_E2E_08528, RS_E2E_08548)

8.3.12 E2E Profile 44 routines

8.3.12.1 E2E_P44Protect

[SWS_E2E_91027]

Service Name	E2E_P44Protect	
Syntax	<pre>Std_ReturnType E2E_P44Protect (const E2E_P44ConfigType* ConfigPtr, E2E_P44ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x50	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 44. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

]()

8.3.12.2 E2E_P44ProtectInit

[SWS_E2E_91028]

Service Name	E2E_P44ProtectInit	
Syntax	<pre>Std_ReturnType E2E_P44ProtectInit (E2E_P44ProtectStateType* StatePtr)</pre>	
Service ID [hex]	0x51	
Sync/Async	Synchronous	

Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the protection state.	
Available via	E2E.h	

]()

[SWS_E2E_10001] In case State is NULL, E2E_P44ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0. (RS_E2E_08539)

8.3.12.3 E2E_P44Forward

[SWS_E2E_91029]{DRAFT} [

Service Name	E2E_P44Forward (draft)	
Syntax	<pre>Std_ReturnType E2E_P44Forward (const E2E_P44ConfigType* ConfigPtr, uint16 Length, E2E_P44CheckStatusType Checkstatus, E2E_P44ProtectStateType StatePtr, uint8* DataPtr)</pre>	
Service ID [hex]	0x52	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	Checkstatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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	
Available via	E2E.h	

]()

8.3.12.4 E2E_P44Check

[SWS_E2E_91030]

Service Name	E2E_P44Check	
Syntax	<pre>Std_ReturnType E2E_P44Check (const E2E_P44ConfigType* ConfigPtr, E2E_P44CheckStateType* StatePtr, const uint8* DataPtr, uint16 Length)</pre>	
Service ID [hex]	0x53	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	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.	
Available via	E2E.h	

]()

8.3.12.5 E2E_P44CheckInit

[SWS_E2E_91031][

Service Name	E2E_P44CheckInit	
Syntax	Std_ReturnType E2E_P44CheckInit (E2E_P44CheckStateType* StatePtr)	
Service ID [hex]	0x55	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the check state.	
Available via	E2E.h	

]()

[SWS_E2E_10002] In case State is NULL, E2E_P44CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF.
2. Status to E2E_P44STATUS_ERROR. (RS_E2E_08528, RS_E2E_08539)

8.3.12.6 E2E_P44MapStatusToSM

[SWS_E2E_91032][

Service Name	E2E_P44MapStatusToSM	
Syntax	E2E_P44CheckStatusType E2E_P44MapStatusToSM (Std_ReturnType CheckReturn, E2E_P44CheckStatusType Status)	
Service ID [hex]	0x56	

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P44Check function
	Status	Status determined by E2E_P44Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_P44CheckStatus- Type	Profile-independent status of the reception on one single Data in one cycle.
Description	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.	
Available via	E2E.h	

l()

[SWS_E2E_10003] If CheckReturn = E2E_E_OK, then the function
E2E_P44MapStatusToSMshall return the values depending on the value of Status:

Status	Return value
E2E_P44STATUS_OK or E2E_P44STATUS_OKSOMELOST	E2E_P_OK
E2E_P44STATUS_ERROR	E2E_P_ERROR
E2E_P44STATUS_REPEATED	E2E_P_REPEATED
E2E_P44STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P44STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

If CheckReturn != E2E_E_OK, then the function E2E_P44MapStatusToSM() shall return
E2E_P_ERROR (regardless of value of Status).

_(RS_E2E_08548)

8.3.13 E2E State machine routines

8.3.13.1 E2E_SMCheck

[SWS_E2E_00340]

Service Name	E2E_SMCheck	
Syntax	<pre>Std_ReturnType E2E_SMCheck (E2E_PCheckStatusType ProfileStatus, const E2E_SMConfigType* ConfigPtr, E2E_SMCheckStateType* StatePtr)</pre>	
Service ID [hex]	0x30	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ProfileStatus	Profile-independent status of the reception on one single Data in one cycle
	ConfigPtr	Pointer to static configuration.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK E2E_E_WRONGSTATE For definitions for return values, see SWS_E2E_00047.
Description	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_POXCheck() function.	
Available via	E2E.h	

](RS_E2E_08548)

[SWS_E2E_00371] In case State is NULL or Config is NULL, the function E2E_SMCheck shall return immediately with E2E_E_INPUTERR_NULL.

Else, the function E2E_SMCheck shall perform the logic according to the specified state machine.](RS_E2E_08528)

8.3.13.2 E2E_SMCheckInit

[SWS_E2E_00353]

Service Name	E2E_SMCheckInit	
Syntax	<pre>Std_ReturnType E2E_SMCheckInit (E2E_SMCheckStateType* StatePtr, const E2E_SMConfigType* ConfigPtr)</pre>	
Service ID [hex]	0x31	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration of the state machine
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description	Initializes the state machine.	
Available via	E2E.h	

](RS_E2E_08528)

[SWS_E2E_00370] In case State is NULL or Config is NULL, the function E2E_SMCheckInit shall return immediately with E2E_E_INPUTERR_NULL.

Else (i.e. both pointers are not NULL), the function E2E_SMCheckInit shall initialize the State structure, setting:

1. ProfileStatusWindow[] to E2E_P_NOTAVAILABLE on each element of the array
2. WindowTopIndex to 0
3. OKCount to 0
4. ERRORCount to 0
5. SMState to E2E_SM_NODATA

and it shall return with E2E_E_OK.](RS_E2E_08528)

8.3.14 Auxiliary Functions

8.3.14.1 E2E_GetVersionInfo

[SWS_E2E_00032]

Service Name	E2E_GetVersionInfo	
Syntax	<pre>void E2E_GetVersionInfo (Std_VersionInfoType* VersionInfo)</pre>	
Service ID [hex]	0x14	
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	E2E.h	

](SRS_BSW_00003)

[SWS_E2E_00033] The function E2E_GetVersionInfo shall return the version information of this module. The version information includes:

- vendor ID
- module ID
- sw_major_version
- sw_minor_version
- sw_patch_version](RS_E2E_08528)

8.4 Call-back notifications

None. The E2E library does not have call-back notifications.

8.5 Scheduled functions

None. The E2E library does not have scheduled functions.

8.6 Expected Interfaces

In this chapter, all interfaces required from other modules are listed. The functions of the E2E Library are not allowed to call any other external functions than the listed below. In particular, E2E library does not call RTE.

[SWS_E2E_00110] The E2E library shall not call any functions from external modules apart from explicitly listed expected interfaces of E2E Library. (RS_E2E_08528)

8.6.1 Mandatory Interfaces

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

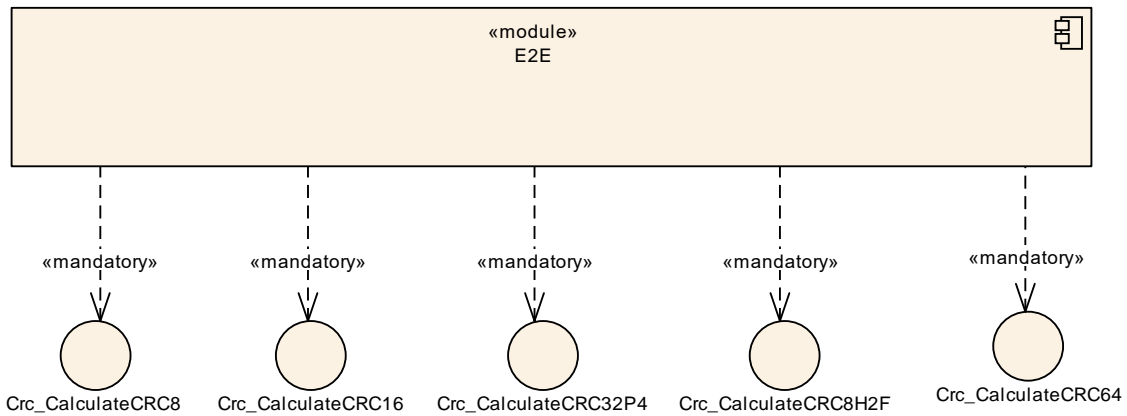


Figure 8-28: Expected mandatory interfaces by E2E library

9 Sequence Diagrams for invoking E2E Library

This chapter describes how the E2E library is supposed to be invoked by the callers. It shows how the E2E Library is used to protect data elements and I-PDUs.

9.1 Sender

[UC_E2E_00202] [During its initialization, the Sender shall instantiate the structures PXXConfigType and PXXProtectStateType, separately for each Data to be protected.] (RS_E2E_08528)

[UC_E2E_00203] [During its initialization, the Sender shall initialize the PXXConfigType with the required configured settings, for each Data to be protected.] (RS_E2E_08528)

Settings for each instance of PXXConfigType are different for each Data; they are defined in Software Component template in the class EndToEndDescription.

[UC_E2E_00204] [During its initialization, the Sender shall initialize the E2E_PXXProtectStateType for each Data, with the configured following values: Counter = 0.] (RS_E2E_08528)

[UC_E2E_00205] [In every send cycle, the Sender shall invoke once the function E2E_PXXProtect() and then once the function to transmit the data (e.g. Rte_Send_<p>_<o>() or PduR_ComTransmit()).

This means that is not allowed e.g. to call E2E_PXXProtect() twice without having Rte_Send_<p>_<o>() in between. It is also not allowed e.g. to call PduR_ComTransmit() twice without having E2E_PXXProtect() in between.] (RS_E2E_08528)

9.1.1 Sender of data elements

The diagram below specifies the overall sequence involving the E2E Library called by the Sender of data elements. The Sender itself can be realized by one or more modules/files. After the diagram, there are requirements specific to Sender of data elements.

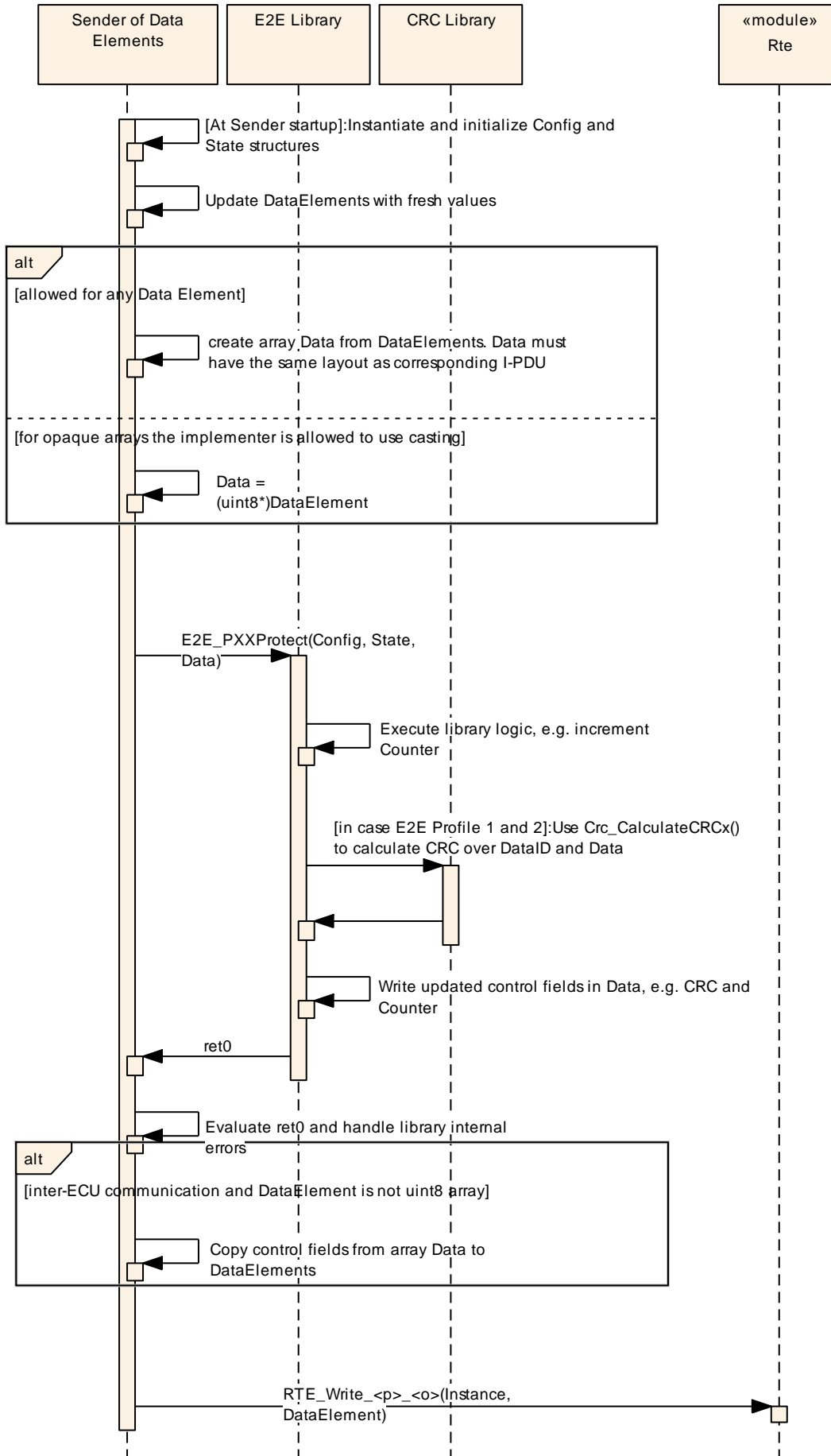


Figure 9-1: Sender of data elements

After the new data element is available, before calling `E2E_PXXProtect()`, the Sender of data elements, shall:

[UC_E2E_00230] [In case the data element communication is inter-ECU and the data element is not an opaque `uint8` array, then the user of the E2E Library shall serialize the data element into the array `Data`. The content of the array `Data` shall be equal to the content of the serialized representation of corresponding signal group in an I-PDU.] (RS_E2E_08528)

Note that there can be several protected signal groups in an I-PDU.

To fulfill the above requirement, the user of E2E library needs to know how safety-related data elements are mapped by RTE to signals and then by COM to areas in I-PDUs so that it can replay this step. This is quite a complex activity because this means that the Sender needs to do a “user-level” COM.

[UC_E2E_00232] [For sending of data elements different from opaque arrays, the caller of E2E Library shall serialize the data element to `Data`, then it shall call the `E2E_PXXProtect()` routine and then it shall copy back the control fields from `Data` to data element.] (RS_E2E_08528)

By its nature, the serialization involves data copying. If a data element is an opaque array, then there is no need for data serialization to array and the caller can cast a data element to `uint8*`. However, to avoid a special treatment of opaque arrays with respect to other data types, an implementer may decide to apply serialization of data element to `Data` also for opaque arrays.

The offsets of control fields in `Data` are defined in Software Component Template metaclass `EndToEndDescription`.

9.1.2 Sender at signal group level

The diagram below species the overall sequence involving the E2E Library by the Sender at the signal group level. The Sender itself can be realized by one or more modules/files (e.g. COM plus callouts, or COM plus complex device driver).

The diagram shows the example when there is only one E2E-protected signal group in the I-PDU, but in general it is possible to have several of them (0 or 1 E2E-protections per signal group). In such case, the sender of I-PDUs invokes `E2E_PXXProtect` on each E2E-protected signal group.

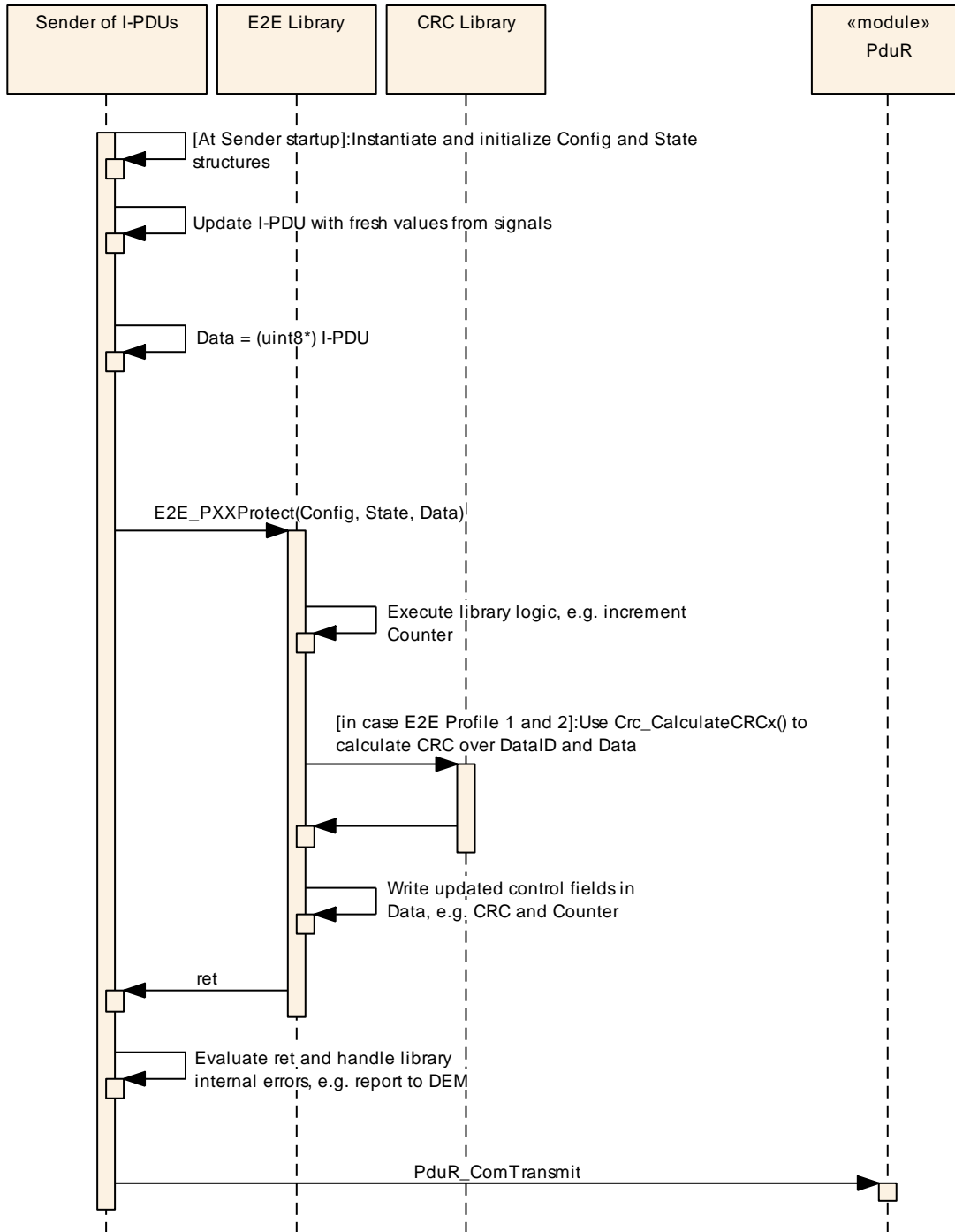


Figure 9-2: Sender of I-PDUs

9.2 Receiver

[UC_E2E_00206][During its initialization, the Receiver shall instantiate the structures PXXConfigType and PXXReceiverType.](RS_E2E_08528)

Note: When selecting the following initialization and configuration parameters the functional behaviour of the enhanced E2E_PXXCheck()-functions (introduced in

AUTOSAR R4.0.4 and R3.2.2) is application-wise backward compatible to the E2E_PxxCheck()-function of the earlier AUTOSAR releases:

- State → SyncCounter := 0;
- Config → MaxNoNewOrRepeatedData := 14 (when using Profile 1);
- Config → MaxNoNewOrRepeatedData := 15 (when using Profile 2);
- Config → SyncCounterInit := 0;

Exemplary configuration parameters and resulting behaviour of the E2E_PxxCheck function:

E2E_PxxConfigType:

- Config → MaxDeltaCounterInit = 2 (i.e. tolerance interval for initial counter differences)
- Config → MaxNoNewOrRepeatedData= 3 (i.e. tolerance interval for maximum counter differences)
- Config → SyncCounterInit = 2 (i.e. duration of counter continuity check)

Timeout interval checked by SWC = 8 transmission cycles

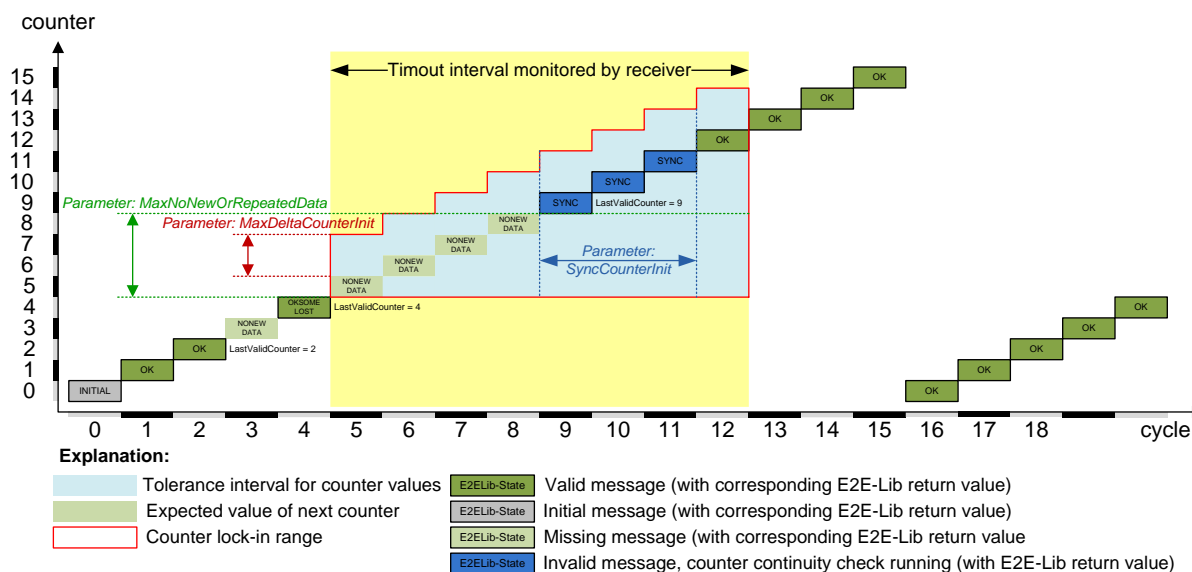


Figure 9-3: Configuration parameters of the E2E_PxxCheck() function and their effects

Clarification regarding SYNC states in Figure 9-3: In cycle 9, the counter value is not trustable anymore since the NoNewOrRepeatedData exceeds MaxNoNewOrRepeatedData. The resulting behavior is similar to as if an "unexpected behavior of the counter" is detected in cycle 9. Thus, the "counter continuity check" spans from cycle 10-11.

[UC_E2E_00207] During its initialization, the Receiver shall initialize the PXXConfigType with the required configured settings, for each Data. (RS_E2E_08528)

Settings for each instance of PXXConfigType are different for each Data; they are defined in Software Component template in the class EndToEndDescription.

[UC_E2E_00209] In every receive cycle, the Receiver shall:

1. Invoke once the reception function Rte_Read_<p><o>().
2. Set the attribute State->NewDataAvailable to TRUE if new data has been received without any errors:
 - a. In case of single channel or channel 1: State->NewDataAvailable = (retRteRead == RTE_E_OK) ? TRUE : FALSE;
 - b. In case of channel 2: State->NewDataAvailable = TRUE; (note: the second channel has no access to Rte_Read return value).
3. Update Data, using received data element or I-PDU.

4. Call once the function E2E_PXXCheck().
5. Handle results (return value and State parameter) returned by E2E_PXXCheck().](RS_E2E_08528)

Note: In case of single channel only, the NewDataAvailable flag may additionally incorporate the return value of the Rte_IsUpdated() API (if available) in the following way:

1. Invoke once the function Rte_IsUpdated_<p>_<o>().
2. Distinguish
 - a) If Rte_IsUpdated_<p>_<o>() returned FALSE : Set the attribute State->NewDataAvailable to FALSE and retRteRead to RTE_E_OK
 - b) If Rte_IsUpdated_<p>_<o>() returned TRUE :
 - i. Invoke once the reception function Rte_Read_<p>_<o>()
 - ii. Set the attribute State->NewDataAvailable to TRUE if Rte_Read_<p>_<o>() returned RTE_E_OK, otherwise set it to FALSE
3. Steps 3.-5. as stated in [UC_E2E_00209].

This resembles the optional functionality of E2EPW_Read_<p>_<o>() as specified in AR 3.2.1 – 3.2.2 / AR 4.0.1 – AR 4.1.1. It was changed as the functionality of Rte_IsUpdated_<p>_<o>() strongly depends on the underlying Com stack to provide a reliable reception indication (callback). Otherwise, corrupted data might be masked.

The Functions E2E_PXXCheck() return the results of verification, by means of parameter State. Within the State (structure E2E_PXXCheckStateType), there is the attribute LostData, which is has a defined value and makes sense only for the following states: E2E_PXXSTATUS_OK and E2E_PXXSTATUS_OKSOMELOST.

[UC_E2E_00233] If the return from the function E2E_PXXCheck() is different than E2E_PXXSTATUS_OK and E2E_PXXSTATUS_OKSOMELOST, then the caller shall not evaluate the attribute State->LostData.](RS_E2E_08528)

9.2.1 Receiver at data element level

The diagram below species the overall sequence involving the E2E Library called by the Receiver at data element level. The Sender itself can be realized by one or more modules/files. After the diagram, there are requirements specific to Sender of data elements.

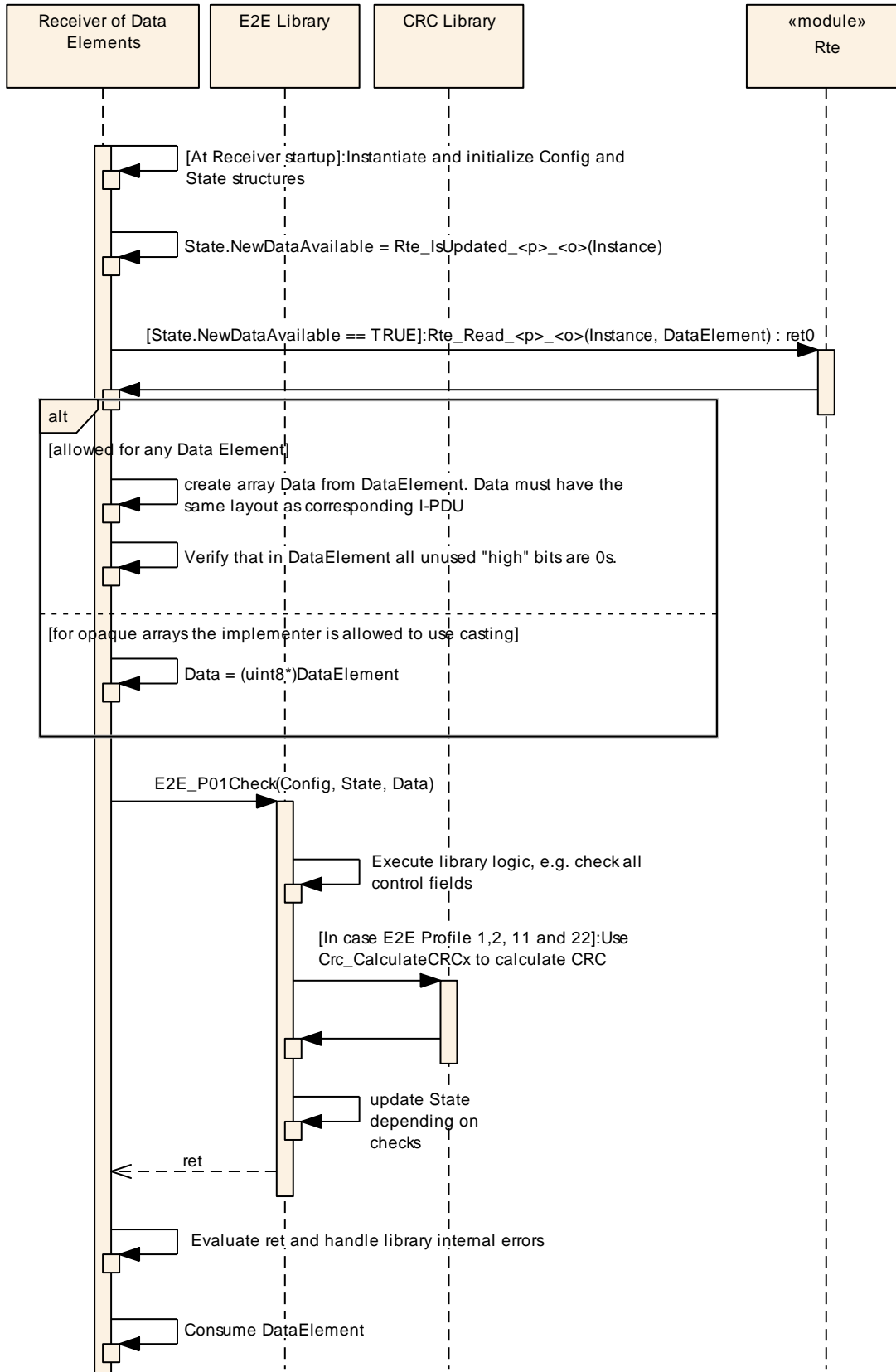


Figure 9-4: Receiver of data elements

[UC_E2E_00277] [In case the data element communication is inter-ECU and the data element is not an opaque uint8 array, then the Receiver shall serialize the data element into the array Data. The layout (content) of Data shall be the same as the layout of the

corresponding I-PDU over which the data element is sent. Moreover, the Receiver shall also verify that all bits that are not transmitted in I-PDU (i.e. which are not present in Data) are equal to 0.](RS_E2E_08528)

To fulfill the above requirement, the Receiver needs to know how safety-related data elements are mapped by RTE to signals and then by COM to I-PDUs so that it can replay this step. This is quite a complex activity because this means that the Sender needs to do a “user-level” COM.

An example of bit verification: Assuming that 10 bits in I-PDU are expanded by COM into 16-bit signal and then by RTE into a 16-bit data element. In this case, the 6 most significant bits of the data element shall be 0. This shall be verified by the Receiver.

[UC_E2E_00278]For reception of data elements different from opaque arrays, the caller of E2E Library shall serialize the data element to Data, then it shall call the check routine.](RS_E2E_08528)

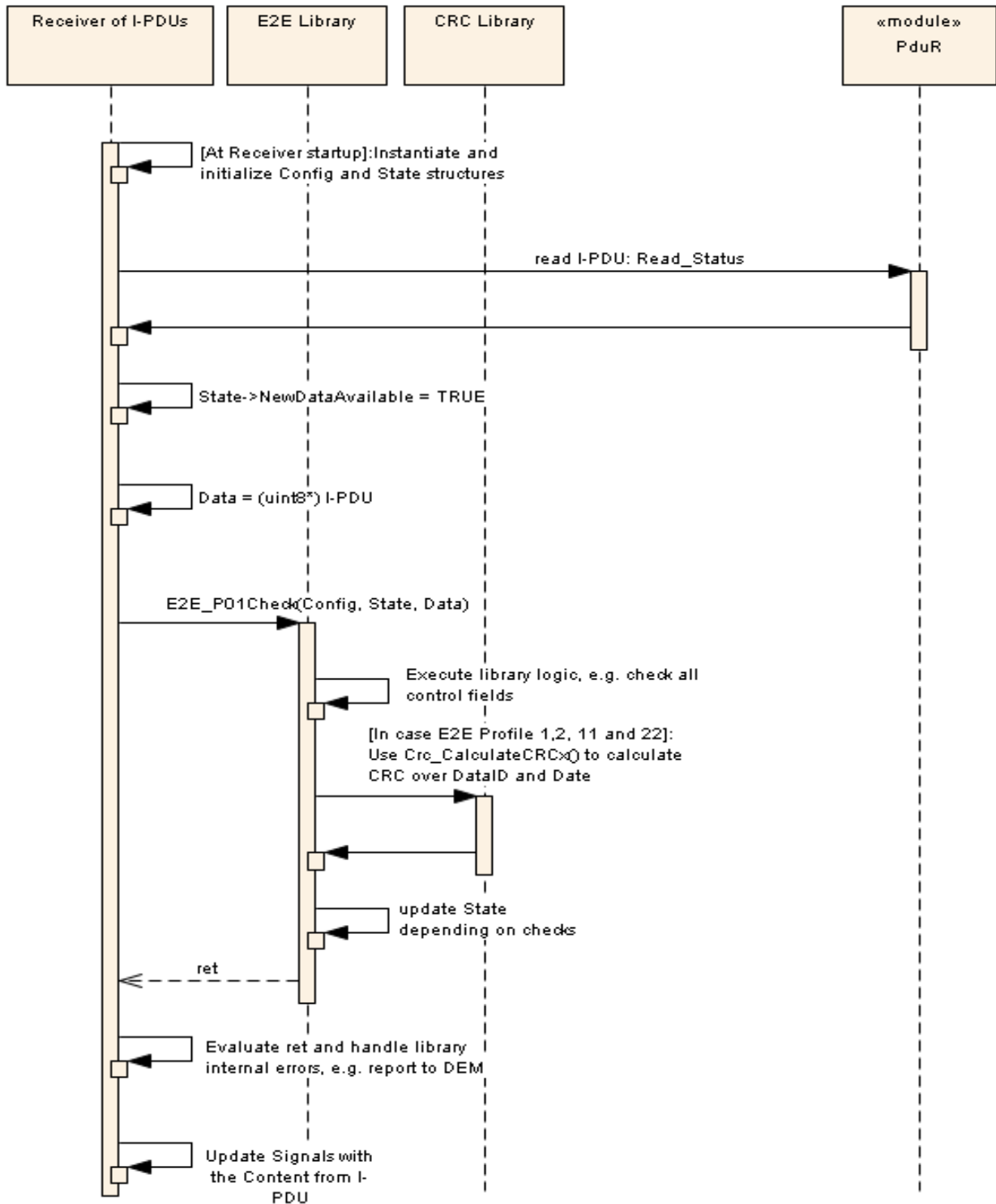
9.2.2 Receiver at signal group level

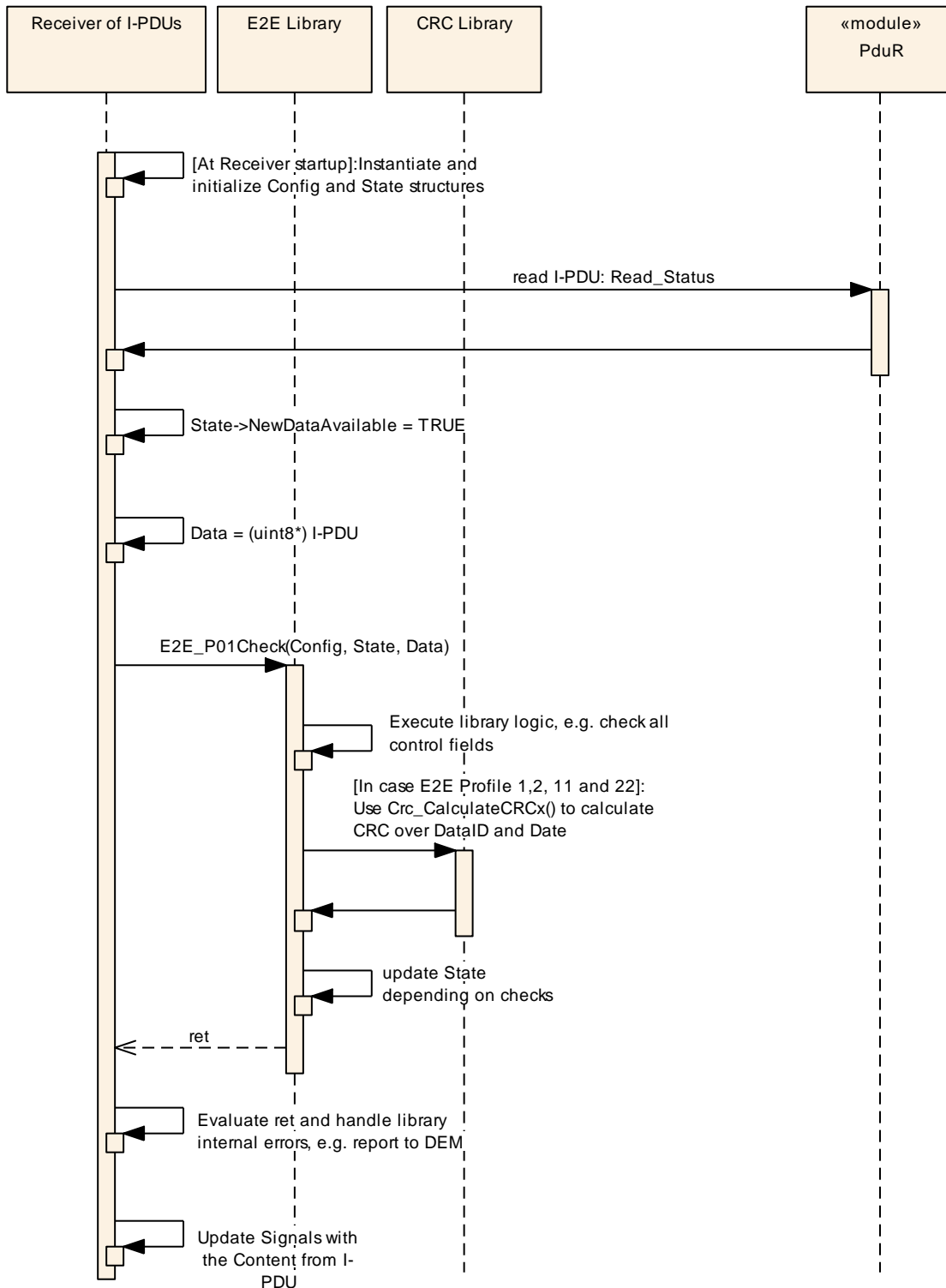
The diagram below summarizes the sequence involving the E2E Library by the Receiver at signal group level.

The diagram shows the example when there is only one E2E-protected signal group in the I-PDU, but in general, it is possible to have several of them (0 or 1 E2E-protections per signal group). In such case, the receiver of I-PDUs invokes E2E_PXXCheck on each E2E-protected signal group.

Diagram below shows the step "State."

This applies only for channel 2. For channel 1 and single channel, the step is "State.NewDataAvailable = (ret0 == RTE_E_OK) ? TRUE : FALSE".





Figure

9-5: Receiver of I-PDUs

10 Configuration specification

E2E Library, like all AUTOSAR libraries, has no configuration options. All the information needed for execution of Library functions is passed at runtime by function parameters. For the functions E2E_PXXProtect() and E2E_PXXCheck(), one of the parameters is Config, which contains the options for the protection of Data.

[SWS_E2E_00037]「The E2E library shall not have any configuration options.」

(SRS_BSW_00344, SRS_BSW_00345, SRS_BSW_00159, SRS_BSW_00167,
SRS_BSW_00171, SRS_BSW_00170, SRS_BSW_00101)

10.1 Published Information

[SWS_E2E_00038]「The standardized common published parameters as required by SRS_BSW_00402 in the General Requirements on Basic Software Modules[3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1].」(SRS_BSW_00004)

Additional module-specific published parameters are listed below if applicable.

11 Annex A: Safety Manual for usage of E2E Library

This chapter contains requirements on usage of E2E Library when designing and implementing safety-related systems, which are depending on E2E Protection of communication.

The description how to invoke/call of E2E Library API is defined in Chapter 9.

11.1 E2E profiles and their standard variants

E2E Library provides two E2E Profiles. They can be used for inter and intra ECU communication.

Because E2E Profile 1 has several configuration options, the recommended/default values for the options are defined as standard E2E profile 1 variants.

[UC_E2E_00053] Any user of E2E Profile 1 shall use whenever possible the defined E2E variants. (RS_E2E_08528)

11.2 E2E error handling

The E2E library itself does not handle detected communication errors. It only detects such errors for single received data elements and returns this information to the callers (e.g. SW-Cs), which have to react appropriately.

A general standardization of the error handling of an application is usually not possible.

[UC_E2E_00235] The user (caller) of E2E Library, in particular the receiver, shall provide the error handling mechanisms for the faults detected by the E2E Library. (RS_E2E_08528)

11.3 Methodology of usage of E2E Library

This section summarizes the steps needed to use the E2E Library. In AUTOSAR R4.0 the usage of E2E Library is not defined by AUTOSAR methodology. There are four main steps, as described below.

In the first step, the user selects the architectural approach how E2E Library is used in a given system (through COM callouts, through E2E Protection wrapper etc). There are several architectural solutions of usage of E2E Library described in Chapter 11.5.

In the second step, the user selects which data elements or signal groups need to be protected and with which E2E Profile. In principle, all transmitted data identified as safety-related are those that need to be protected.

In the third step, the user determines the settings for each selected data element or signal group to be protected. The settings are stored in Software Component Template metaclass `EndToEndDescription`. The settings include e.g. Data ID, CRC offset.

1. For each signal group to be protected, there is a separate instance of `EndToEndDescription`, associated in System Template to `ISignallPdu` metaclass.
2. For each data element to be protected, there is a separate instance of `EndToEndDescription`, associated indirectly to `VariableDataPrototype`, `SenderComSpec` and `ReceiverComSpec` metaclasses.

In the fourth and last step, the user generates (or otherwise develops) the necessary glue code (e.g. E2E Protection Wrapper, COM callouts), responsible for invocation of E2E Library functions. The glue code serves as an adapter between the communication modules (e.g. COM, RTE) and E2E Library.

11.4 RTE configuration constraints for SW-C level protection

In case the E2E Library is used to protect data elements, there are a few constraints how RTE needs to be configured.

If the protection takes place at the level of I-PDUs, then there are no constraints from the side of E2E on RTE configuration.

11.4.1 Communication model for SW-C level protection

AUTOSAR RTE supports different communication models, like client-server, sender-receiver, mode switch etc.

11.4.2 Multiplicities for SW-C level protection

The E2E Library is not intended to be used for N:1 sender-receiver multiplicities.

[UC_E2E_00258] In case the E2E Library is used to protect data elements, then the selected multiplicity shall be 1:N or 1:1. (RS_E2E_08528)

11.4.3 Explicit access

Sender-receiver SW-C communication is asynchronous in the sense that the sender does not wait for the receiver. It means that the sender passes the data element to RTE and continues the execution – it does not wait for the receiver to receive the data – this is not configurable. RTE transmits the data to the receiver concurrently to the execution of the sender.

Now, the question is how the receiver gets the data. There are two ways to do it in AUTOSAR, which is configurable in RTE:

1. The receiver waits for new data: it is blocked/waiting until new data element from the sender arrives (RTE communication modes “wake up of wait point” and “activation of Runnable entity”)
2. The receiver gets the currently available data element from RTE, i.e. the most recent data element (RTE communication modes “Implicit data read access” and “Explicit data read access”)

E2E Profile 1 and 2 together with the proposed E2E protection wrapper provide timeout detection (which is one of the failure modes to handle – e.g. message loss). This is achieved by having the receiver executing independently from the reception of the data, and by the usage of a counter within E2E Profiles. By this means, if e.g. a data element is lost, it is seen by the receiver that every time the read data element has the same counter. This however requires that the receiver is not solely executed upon the arrival of data.

In case the receiver is event-driven, then a timeout mechanism at the receiver needs to be used. The timeout mechanism is not a part of E2E Library.

[UC_E2E_00089][In case the E2E Library is used to protect data elements, data elements accessed with E2E Protection Wrapper shall use the activation “Explicit data read access” (i.e. it shall not use the activations “Implicit data read access”).](RS_E2E_08528)

11.5 Restrictions on the use of COM features

The following table lists COM features with a brief description and provides a classification of restriction of use in combination with End-to-End communication protection as described in this document.

Note: This list only covers features of the BSW module COM in combination with E2E Library and E2E Protection Wrapper. It does not address features of above layers (e.g. RTE) or use-cases where the E2E Transformer is used. The latter usually is used above the BSW module LdCom.

The restriction classes are as follows:

- **"supported"** means that both (E2E COM Callout and E2EPW) do support this feature.
- **"use case dependent"** means that the feature might be used/usable depending on the actual use case and configuration on sender and receiver side. However, suitability for an actual system and its influence on the safety requirements has to be analysed.
- **"not supported"** means that at least one variant (either E2E COM Callout or E2EPW) does not support this feature or a failure mode can be masked.

COM Feature / brief description	Classification
[SRS_Com_02078] Support of endianness conversion	supported

COM Feature / brief description	Classification
[SRS_Com_02086] Support of Sign-Extension for received signals	supported
[SRS_Com_02042] Initialization of unused areas/ bits of an I-PDU	supported
[SRS_Com_02083] Transmission Modes	use case dependent
[SRS_Com_02082] Two different Transmission Modes	use case dependent
[SRS_Com_02084] Signal data based selection of Transmission Mode	use case dependent
[SRS_Com_02113] Signal data based transmission modes for configured serialized data	use case dependent
[SRS_Com_02046] Configuration of signal notification	supported
[SRS_Com_02080] Cancelation outstanding repetitions in case of a new send request	use case dependent
[SRS_Com_02089] Two configurable options to handle signal timeouts on receiver side	use case dependent
[SRS_Com_02077] Signal invalidation mechanism on sender-side	use case dependent
[SRS_Com_02079] Signal invalidation mechanism on receiver-side	use case dependent
[SRS_Com_02087] Substitution of invalid value by configurable data value	use case dependent
[SRS_Com_02088] Substitution of the last received value by the init value in case of signal timeout	use case dependent
[SRS_Com_00218] Starting/ Stopping communication of I-PDU groups	supported
[SRS_Com_00192] Enabling/ disabling reception deadline monitoring of I-PDU groups	use case dependent
[SRS_Com_02041] Consistent transfer of complex data types	supported
[SRS_Com_02091] Placement of large or dynamical length signals	not supported
[SRS_Com_02092] Support only one dynamic length signal per I-PDU	not supported
[SRS_Com_02093] Dynamic length signal must be placed last in I PDU	not supported
[SRS_Com_02094] Dynamic length signals must be of type UINT8[n]	not supported
[SRS_Com_02095] TP shall be used to fragment and reassemble large signals and dynamical signals	not supported
[SRS_Com_02030] Identify if a signal/signal group is updated by the sender	use case dependent
[SRS_Com_02058] Deadline monitoring of receiving updated signals/signal groups	use case dependent
[SRS_Com_02099] I-PDU Counter mechanism	use case dependent
[SRS_Com_02100] I-PDU Counter configuration	use case dependent
[SRS_Com_02101] Transmission and reception using I-PDU Counter	use case dependent
[SRS_Com_02102] I-PDU Counter error handling	use case dependent
[SRS_Com_02103] I-PDU Replication mechanism	use case dependent
[SRS_Com_02104] I-PDU replication configuration	use case dependent
[SRS_Com_02105] Transmission and reception using I-PDU Replication	use case dependent
[SRS_Com_02106] I-PDU Replication error handling	use case dependent
Minimum Delay Time	use case dependent
Filtering at receiver side (e.g. COM273)	use case dependent
Filtering at sender side	use case dependent
Multiple Signal groups within an I-PDU	use case dependent

Table 11-1: Classification of COM features

12 Annex B: Application hints on usage of E2E Library

To enable the proper usage of the E2E Library different solutions are possible. They may depend e.g. on the integrity of RTE, COM or other basic software modules as well as the usage of other SW/HW mechanisms.

The user is responsible for selecting the solution for usage of E2E Library that is fulfilling safety requirements of his particular safety-related system.

Each particular implementation based on solutions described in this chapter needs to be evaluated with regard to functional safety prior to their use.

The E2E Library can be used in different ways (each explained in a separate section of this chapter):

1. E2E Protection Wrapper – non-standard integrator software to protect data, above RTE (section 12.1)
2. COM callouts – non-standard integrator code to protect I-PDUs (section 12.2).
3. hybrid / unused (section 12.3)
4. Out-of-box protection at RTE level (E2E Transformer)(section 12.4)

It is also possible to have mixed scenarios, e.g.:

1. For a particular data element, a sender using E2E Protection Wrapper and receiver using COM E2E callouts (or reverse)
2. In a given ECU network or one ECU: some data elements protected with E2E protection Wrapper and some with COM E2E callouts.

The first scenario is useful for network diagnostic (e.g. when a monitoring device without RTE checks messages), or when one of the communication partners does not have RTE.

The best situation is when the integrity of operation of RTE and COM for transmitting/converting safety-related data can be guaranteed. In short, we call this safe RTE and safe COM.

This annex describes two exemplary, basic solutions how E2E Library can be invoked. First, this is by means of a dedicated sub-layer for a SW-C or several SW-Cs (which is called E2E Protection Wrapper, see Chapter 12.1). Secondly, this can be done by means of dedicated COM Callouts invoking E2E Library to protect signal groups representing data elements (which is called COM E2E Callouts, see Chapter 12.2).

Chapter 12.3 shows how a component which requires the Protection Wrapper interfaces (Chapter 12.1) can be integrated on a ECU providing the COM Callout solution (Chapter 12.2).

All necessary options, enabling to generate the code for the described solutions are available in AUTOSAR configuration, defined in System Template [9] and Software Component Template [8]. This contains e.g. association of I-PDUs with Data IDs.

To generate the wrapper, the user defines EndToEnd* metaclasses and associates them to VariableDataPrototypes (representing complex data elements). To generate the COM

E2E callouts for an I-PDU, the user defines EndToEnd* metaclasses and associates them to ISignallPdu metaclass (representing the I-PDU).

There are a few E2E mechanisms in which an I-PDU can be protected. There is a new standard mechanism: E2E Transformer, and there are two de-facto-standard mechanisms COM E2E callouts and E2E Protection Wrapper. Finally, some integrators use their own mechanisms like safe COM module. It makes only sense to use one of the mechanism for a given I-PDU.

[UC_E2E_00271] [A given I-PDU, if protected by E2E, shall be protected by only one E2E mechanism.] (RS_E2E_08528)

12.1 E2E Protection Wrapper

Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach.

In this approach, every safety-related SW-C has its own additional sub-layer (which is a .h/.c file pair) called E2E Protection Wrapper, which is responsible for marshalling of complex data elements into the layout identical to the corresponding I-PDUs (for inter-ECU communication), and for correct invocation of E2E Library and of RTE.

The usage of E2E Protection Wrapper allows the use of VFB communication between SW-Cs¹, without the need of further measures to ensure VFB's integrity.

The communication between such SW-Cs can be within an ECU (which means on the same or different cores or within the same or different memory partitions of a microcontroller) or across ECUs (SW-Cs connected by a VFB also using a network).

The end-to-end protection is a systematic solution for protecting SW-C communication, regardless of the communication resources used (e.g. COM and network, OS/IOC or internal communication within the RTE). Relocation of SW-Cs may only require selection of other protection parameters, but no changes on SW-C application code.

The usage of E2E Protection Wrapper can be optimized by appropriate software/memory partitioning.

The E2E Protection Wrapper does not support multiple instantiation of the SW-Cs. This means, if an SW-C is supposed to use E2E Protection Wrapper, then this SW-C must be single-instantiated.

[UC_E2E_00292] [If the E2E Library is invoked from E2E Protection Wrapper (at the level of data elements), then multiple instantiation is not allowed. For an AUTOSAR software component which uses the E2E Protection Wrapper the value of the attribute

¹The term SW-C includes any software module that has an RTE interface, i.e. a sensor/actuator/application SW-C, an AUTOSAR service, or a Complex Driver.

supportsMultipleInstantiation of the SwcInternalBehavior shall be set to FALSE in the AUTOSAR software component description.

The E2E Protection Wrapper itself is not a part of E2E Library. However, its options are standardized. Most of the options for E2E Protection Wrapper are in System Template [9] and some of them are in Software Component Template [8].](RS_E2E_08528)

[UC_E2E_00249]The integrity of the operation of E2E Protection Wrapper (for transmitting/converting safety-related data) shall be guaranteed.](RS_E2E_08528)

The functions of the E2E Protection Wrapper are not reentrant, therefore they are not to be called concurrently.

[UC_E2E_00288]Each E2E Protection Wrapper function shall not be called concurrently.](RS_E2E_08528)

To implement the above requirement, it is recommended to design the SW-Cs and the E2E ports in the way that one particular E2E Protection Wrapper function is called from one Runnable only, i.e. one E2E Protection Wrapper should “belong” to a particular Runnable.

Note: The caller of E2EPW API functions shall make sure that internal status data structures of E2EPW are initialized correctly. Initialization can be done by ECU start-up code or explicitly via E2EPW init functions.

12.1.1 Functional overview

The E2E Protection Wrapper functions as a wrapper over the Rte_Write and Rte_Read functions, offered to SW-Cs. The E2E Protection Wrapper encapsulates the Rte_Read/Write invocations and protection of data exchange using E2E Library.

For a data element to transmit, there is a set of wrapper functions (Read/Write/Init) generated for Sender and for the Receiver.

The E2E Protection Wrapper functions are responsible for instantiation and initialization of data structures required for calling the E2E Library, for invocation of E2E Library and invocation of Rte_Read/Rte_Write functions and for serialization of data elements. The initialization of data structures depend on specific data element, e.g. the Data ID, or E2E Profile to be used.

The functions E2EPW_Write_<p>_<o>() and E2EPW_Read_<p>_<o>() return 32-bit integers that represent the status.

Figure 12-1 shows the overall flow of usage of E2E Library and E2E Protection Wrapper from SW-Cs (the 1st number on the labels defines the order of execution):

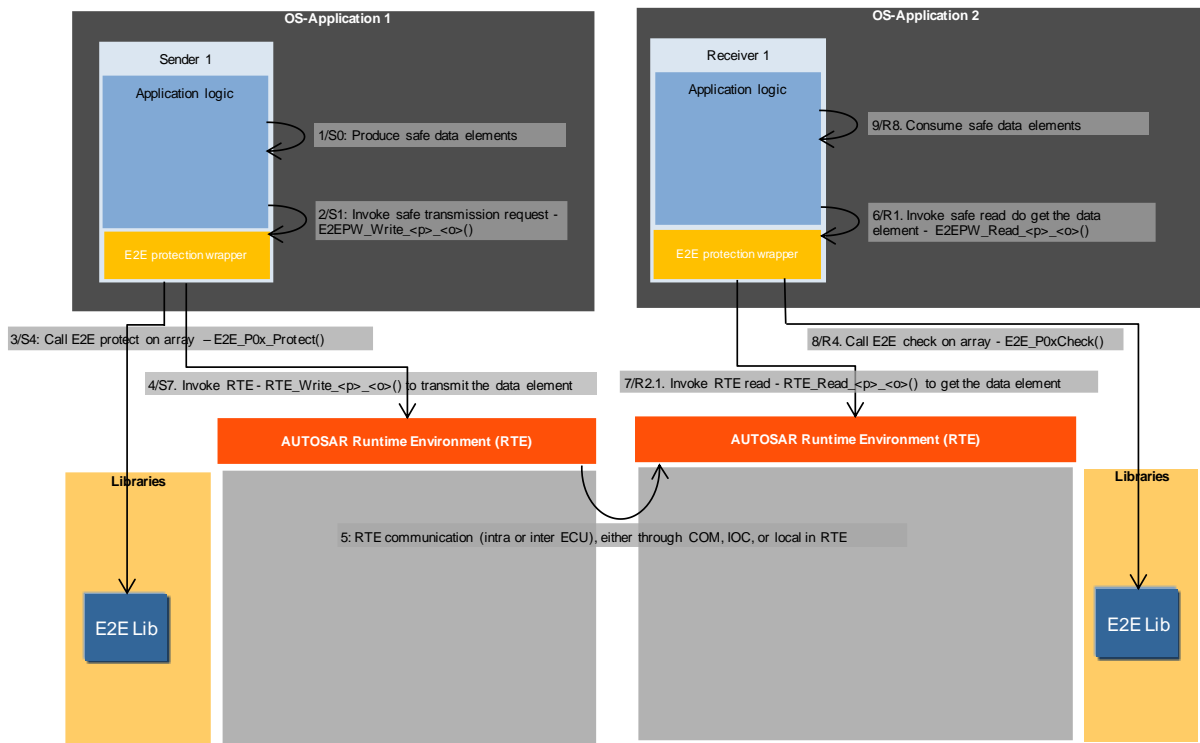


Figure 12-1: Example E2E Protection Wrapper - overall flow

12.1.2 Application scenario with Transmission Manager

It is possible to have one central SW-C to collect safety-related data of several SW-Cs on a given ECU to transmit them combined through a network.

On the sender ECU, there is a dedicated SW-C called Transmission Manager, containing E2E Protection Wrapper. The Transmission Manager collects safety-related data from related SW-Cs, combines them and protects them using E2E Protection Wrapper. Finally, it provides the combined and protected Data as data element to RTE.

On the receiver ECU there may also be a Transmission Manager, which does the reverse steps for the reception of such data.

The Transmission Manager SW-C modules are not part of E2E Library nor part of AUTOSAR.

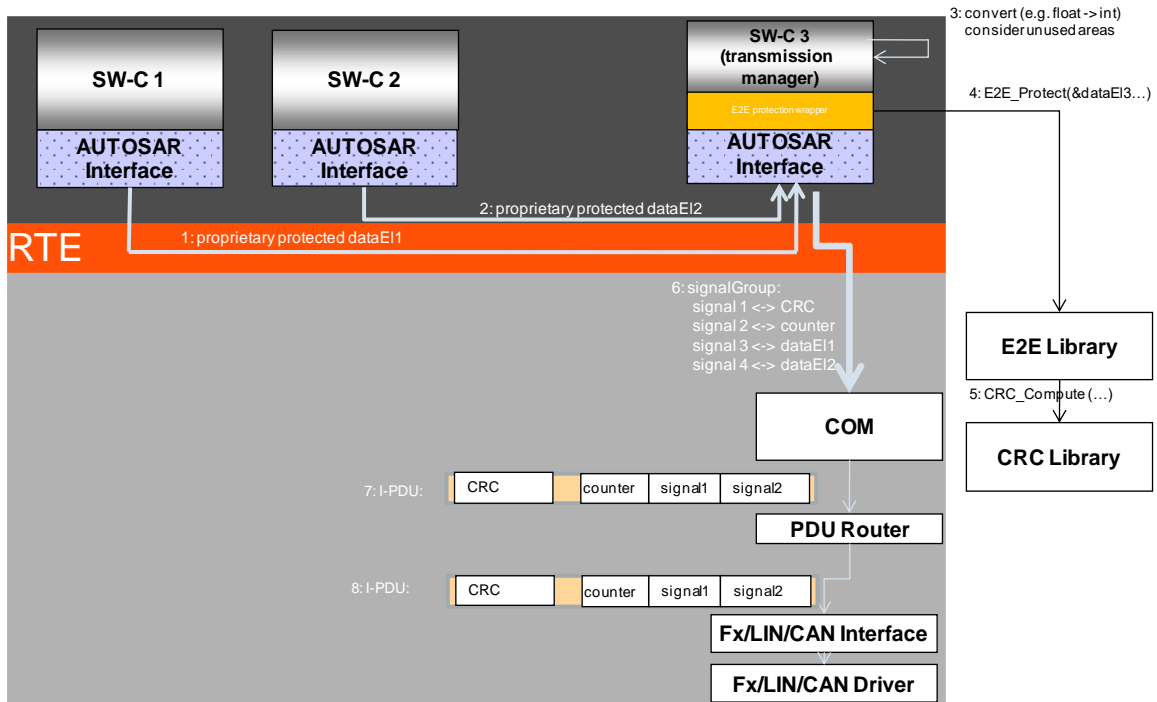


Figure 12-2: Example Transmission Manager – sender ECU

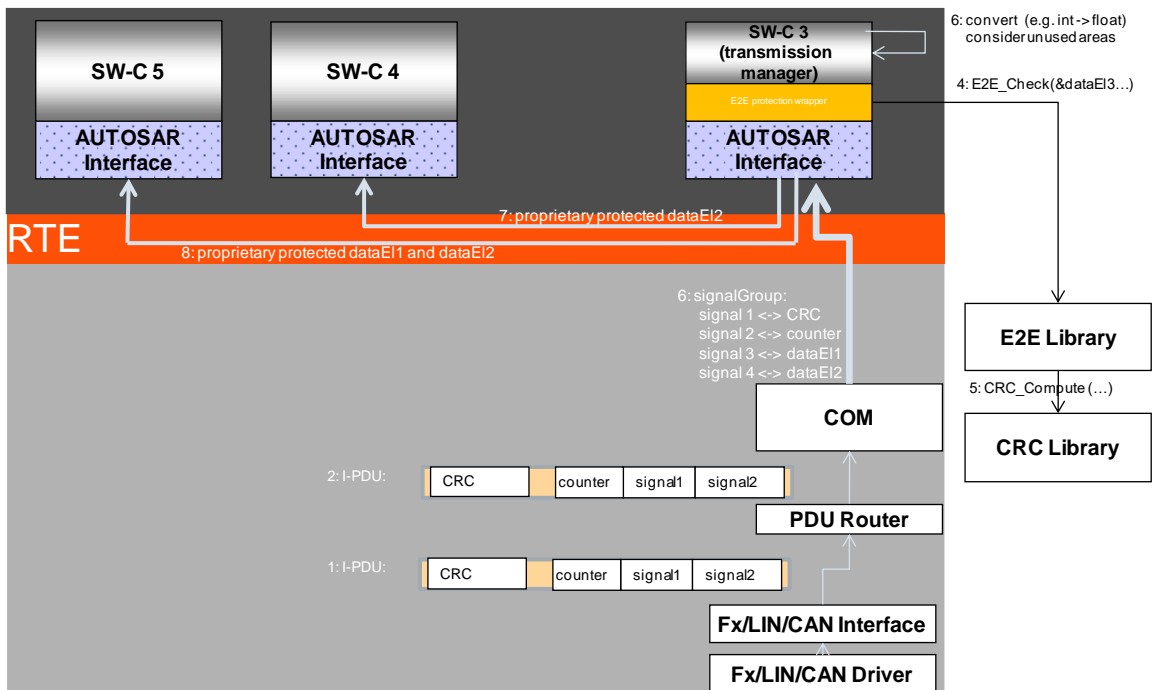


Figure 12-3: Example Transmission Manager – receiver ECU

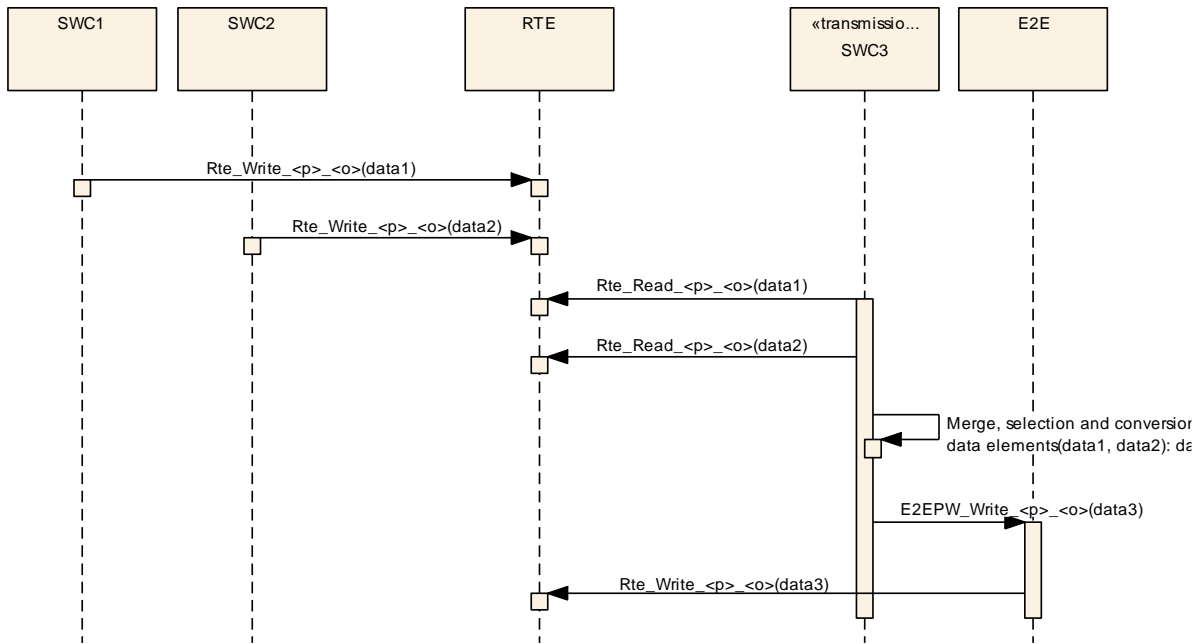


Figure 12-4: Example Transmission Manager –sender ECU sequence

In this example, for SW-C1 and SW-C2 it is not visible that the communication is going through such a Transmission Manager, which can support the portability and optimize resource usage of communication network. It is only through AUTOSAR configuration where it is visible that the receiver of SW-C1 and of SW-C2 is SW-C3.

[UC_E2E_00213] The implementation of the Transmission Manager (as a safety-related Software Component), shall comply with the requirements for the development of safety-related software for automotive domain. (RS_E2E_08528)

12.1.3 Application scenario with E2E Manager and Conversion Manager

This application scenario is similar to the previous one, where the Transmission Manager is split into two separate SW-Cs (E2E Manager and Conversion Manager). The advantage of the scenario is that the E2E Manager can be automatically generated and that Conversion Manager is independent completely from E2E protection.

The Conversion Manager is an SW-C responsible for data conversion, e.g. float-to-integer conversion. On sender ECU, the E2E Manager is responsible for assembling all data elements to be transmitted and protecting them through E2E Protection Wrapper. On receiver ECU, the Conversion Manager is responsible for checking the data through E2E Protection Wrapper and then by filtering out the data that is not needed by receiver Conversion Manager.

The E2E Manager and Conversion Manager SW-C modules are not part of E2E Library nor part of AUTOSAR.

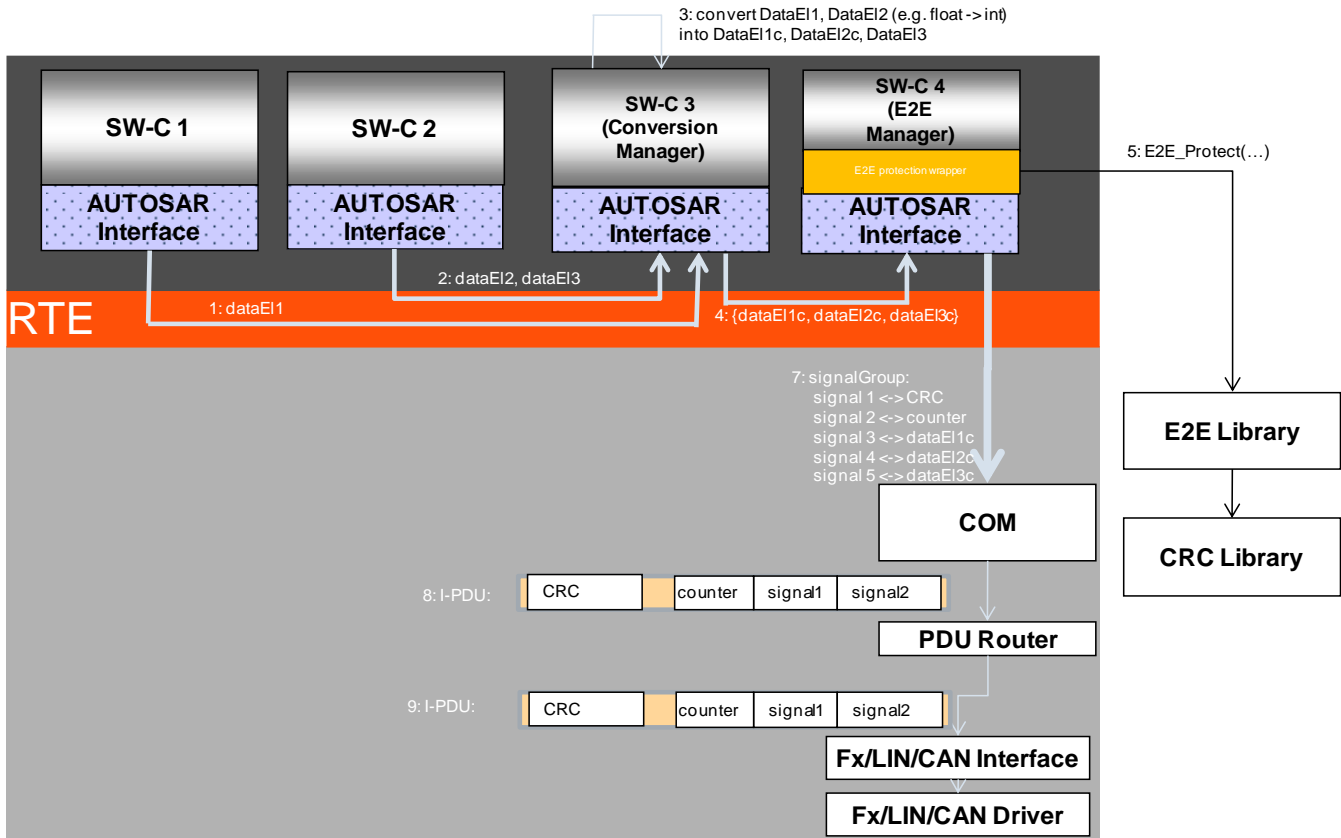


Figure 12-5: E2E Manager and Conversion Manager – sender ECU

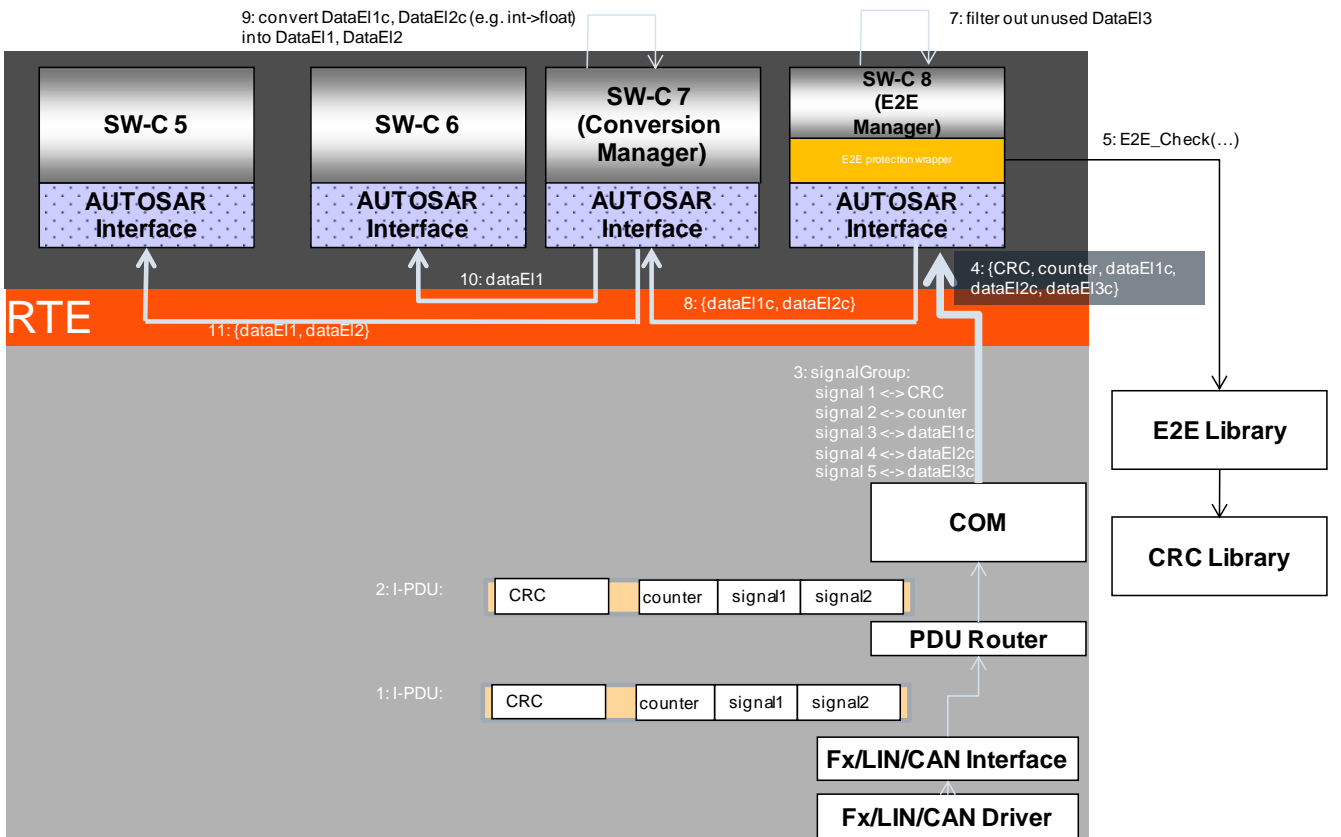


Figure 12-6: E2E Manager and Conversion Manager – receiver ECU

In the above example, the SW-Cs of sender ECU generate three data elements (dataEI1, dataEI2 and dataEI3) but the SW-Cs of receiver ECU use only two data elements

(dataE11 and dataE12). The unused DataE13c is not delivered to Conversion Manager. Thanks to this, if due to e.g. system evolution, the definition of DataE13 changes, then the receiver SW-Cs (SW-C 5, SW-C 6 and SW-C 7 Conversion Manager) do not need to be changed.

The corresponding system configuration description looks as shown by Figure 12-7. Note that the SW-C 7 has as input only the required data elements. The unused data elements (CRC, counter, dataE13c) are not provided:

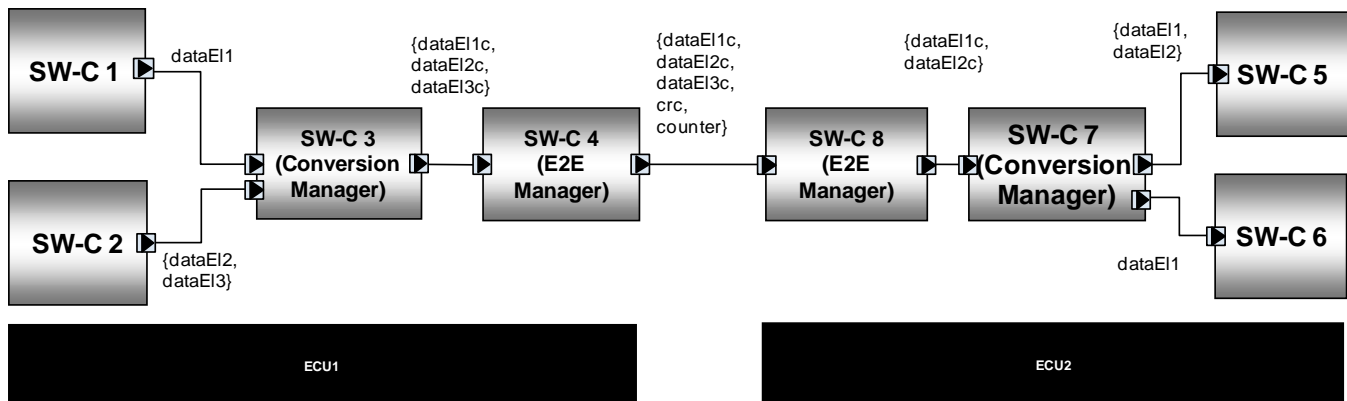


Figure 12-7: E2E Manager and Conversion Manager - system configuration

The E2E protection wrapper of E2E manager can be automatically generated, as described in 0.

The application code of E2E manager is responsible only for “routing” of the input data elements into output data elements, which is also straightforward and can be generated. For the example above, the application code of E2E Manager may look as follows:

```

/* the input complex data element contains primitive data elements
   unused by other SW-Cs of the ECU */
typedef struct {
    uint8 crc;
    uint8 counter;
    uint16 dataE11c;
    uint16 dataE12c;
    uint16 dataE13c;
} Inputswc8Type;

/* the output complex data element is a subset of input, with the
   data used by other SW-Cs of the ECU */
typedef struct {
    uint16 dataE11c;
    uint16 dataE12c;
} Outputswc8DataType;

Inputswc8Type Inputswc8;
Outputswc8Type Outputswc8;
...

/* copy from Inputswc8 the primitive data elements that are also in
   outputswc8 */

```

```
Outputswc8Type.dataE11c = Inputswc8Type.dataE11c;  
Outputswc8Type.dataE12c = Inputswc8Type.dataE12c;
```

[UC_E2E_00274] E2E Manager shall have complex data elements with prefix Input or with prefix Output. There is one-to-one relationship between the data element with input prefix and data element with output prefix. (RS_E2E_08528)

In the example above, there is Inputswc8 and the corresponding Outputswc8.

[UC_E2E_00275] The output data element shall contain the subset of primitive data elements of those of the corresponding input data element (in particular, they may be equal). (RS_E2E_08528)

In the example above, Outputswc8 contains the subset of attributes of Inputswc8. It does not contain dataE13c, crc, nor counter.

For each primitive data element of output complex data element, the (generated) application code of E2E manager shall write it with the value read from the corresponding primitive data element of the input complex data element.

In the example above, the application code of E2E manager copies dataE11c and dataE12c from Inputswc8 to Outputswc8.

[UC_E2E_00272] The implementation of the Conversion Manager and E2E Manager (as a safety-related Software Component), shall comply with the requirements for the development of safety-related software for automotive domain. (RS_E2E_08528)

[UC_E2E_00273] The E2E Manager SW-C at receiver ECU shall filter out the data elements that are not used by the SW-Cs of the ECU. The E2E Manager SW-C at receiver ECU shall forward to Conversion Manager SW-C only the data elements that are used by Conversion Manager SW-C. (RS_E2E_08528)

12.1.4 File structure

[UC_E2E_00239] The E2E Protection Wrapper, for the given SW-C identified with <SWC-Type-short name>, shall be made of two files: E2EPW_<SWC-Type-short name>.c and E2EPW_<SWC-Type-short name>.h. (RS_E2E_08528)

[UC_E2E_00242] The SW-C implementation files that invoke E2E Protection Wrapper functions shall include E2EPW_<SWC-Type-short name>.h. (RS_E2E_08528)

[UC_E2E_00256] The E2E Protection Wrapper shall ensure the integrity of the safety-related data elements. (RS_E2E_08528)

[UC_E2E_00257] The implementation of the E2E Protection Wrapper (as a safety-related Software Component) shall comply with the requirements for the development of safety-related software for the automotive domain. (RS_E2E_08528)

12.1.5 Methodology

Note: Different releases of AUTOSAR have different names for COM classes. The text description below is generalized to fit to different releases, but the diagrams are slightly different (main differences are different names of classes and objects).

During the RTE contract phase (i.e. when SW-C interface files are generated), the standard AUTOSAR RTE generator generates, for an SW-C, the SW-C interface file `Rte_<SWC-Type-short name>.h`. This file contains the RTE's generated functions like `Rte_Write_<p>_<o>()`. For each function in this file used to transmit safety-related data, there is the corresponding function in `Rte_<SWC-Type-short name>.h`.

The E2E protection wrapper can be implemented manually, or can be generated/configured from its description. All necessary information required to generate the E2E Protection Wrapper can be configured using AUTOSAR templates (system template, SW-C template, ECU configuration).

The generation of the E2E protection wrapper can be done along the execution the step "Generate Component API", which step generates "Component API".

[UC_E2E_00248] The E2E Protection Wrapper shall be generated for the complex data elements (represented by `VariableDataPrototype` metaclass) for which the corresponding `EndToEnd*` metaclasses are defined. (RS_E2E_08528)

[UC_E2E_00289] If the E2E Protection is done in the E2E Wrapper then both `EndToEndProtectionISignalPdu` and `EndToEndProtectionVariablePrototype` shall be defined. (RS_E2E_08528)

Most of the settings are defined under Software Component Template [8].

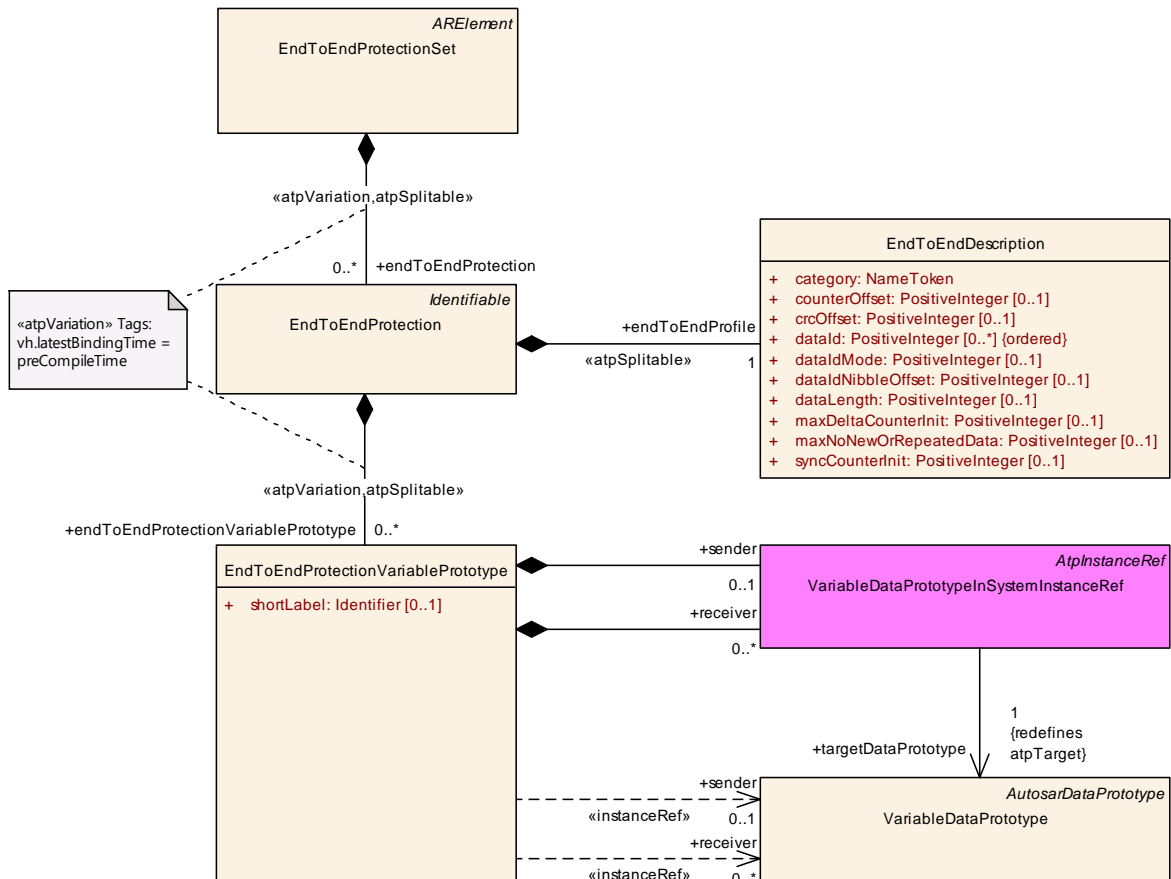


Figure 12-8: Release R4.0.1 and newer: E2E Protection Wrapper configuration (hardcopy from DOC_EndtoEndProtection)

The metaclass `EndToEndProtectionVariablePrototype` defines that a particular (complex) data element shall be protected. This data element has at most one specific sender and any quantity of receivers (`VariableDataPrototype`). The specific settings how the data element shall be protected are defined in the class `EndToEndDescription` (these settings can be reused by different data prototypes).

Apart from configuring `EndToEndProtectionVariablePrototype`, further settings involve the mapping signal groups to I-PDUs, which is done according to System Template [9]:

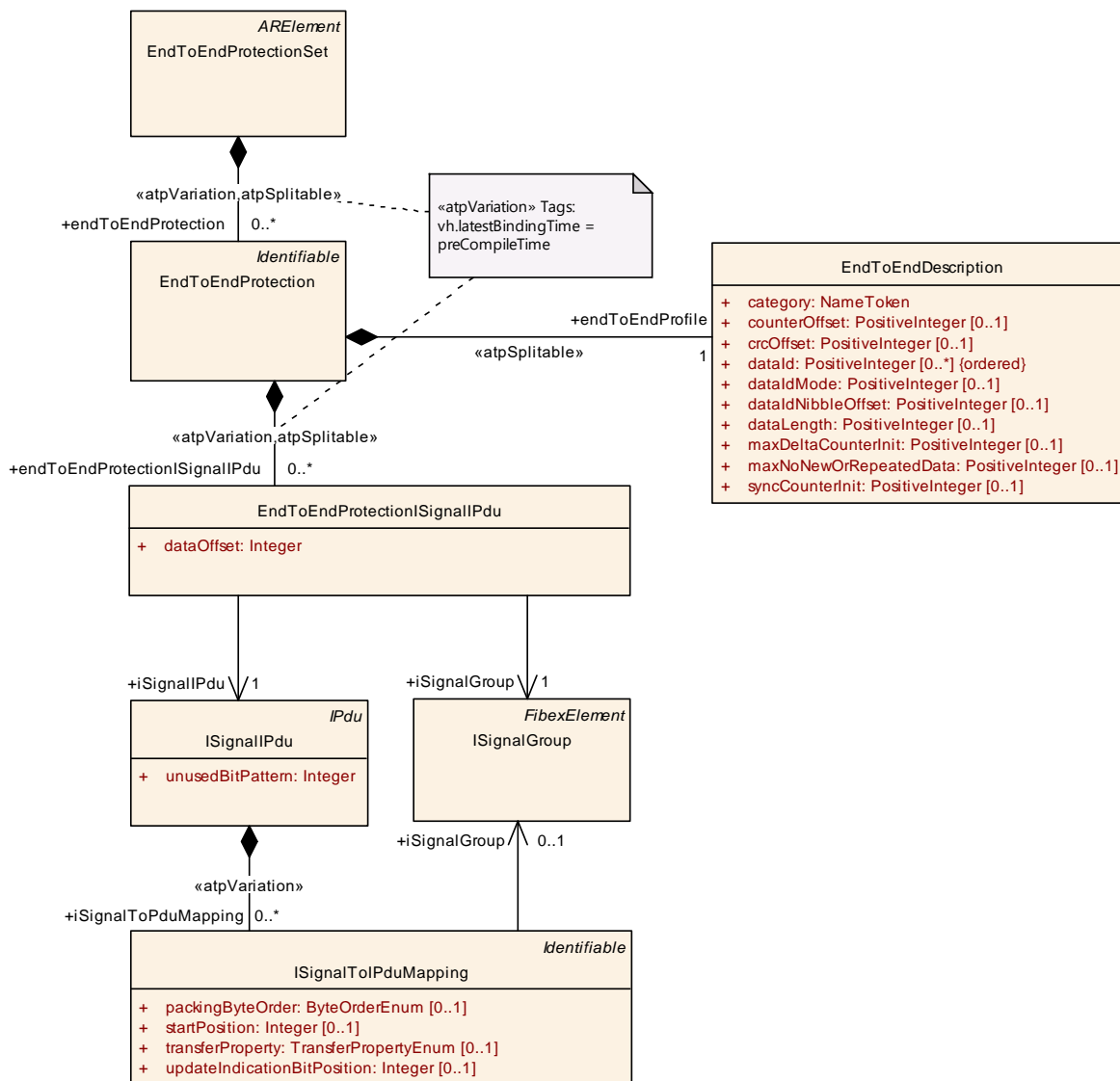


Figure 12-9: Release R4.0.1 and newer: E2E Protection Wrapper configuration (hardcopy from DOC_PduEndToEndProtection)

The important settings are:

1. ISignalIPdu (represents an I-PDU)
 - a. ISignalIPdu.unusedBitPattern: bits that are not used in an I-PDU,
2. ISignalToIPduMapping: describes the mapping of signals to I-PDUs,
 - a. ISignalToIPduMapping.startPosition: offset in bits of a signal in the I-PDU,
3. EndToEndProtectionISignalIPdu: association of one E2E protection to a one I-PDU and to one signal group,
 - a. EndToEndProtectionISignalIPdu.dataOffset: offset in bits of the signal group in the I-PDU.

It is possible to add several signal groups into one I-PDU using several EndToEndProtectionISignalIPdu elements.

The ISignalPdu.unusedBitPattern is used by COM to create the final I-PDU and by E2E Protection Wrapper, to create a correct I-PDU representation of the protected data (on which a correct CRC can be computed).

It is also necessary to configure SenderComSpec and ReceiverComSpec. ReceiverComSpec may override maxDeltaCounterInit provided by EndToEndDescription (by means of attribute ReceiverComSpec. maxDeltaCounterInit). This may be useful if different receivers of one data element (for the same sender) require different settings.

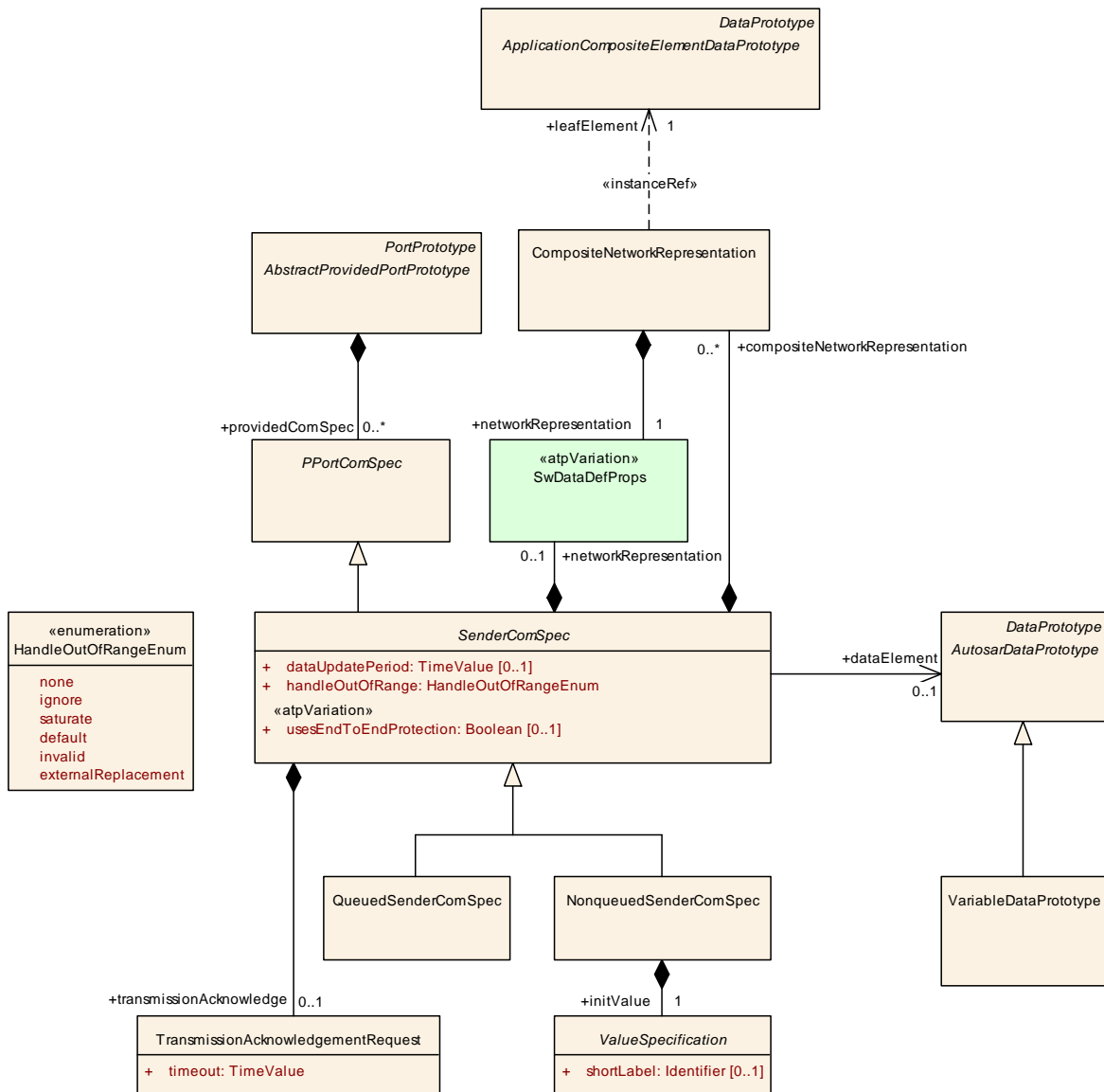


Figure 12-10: Release R4.0.1 and newer: SenderComSpec (hardcopy from DOC_SenderComSpec)

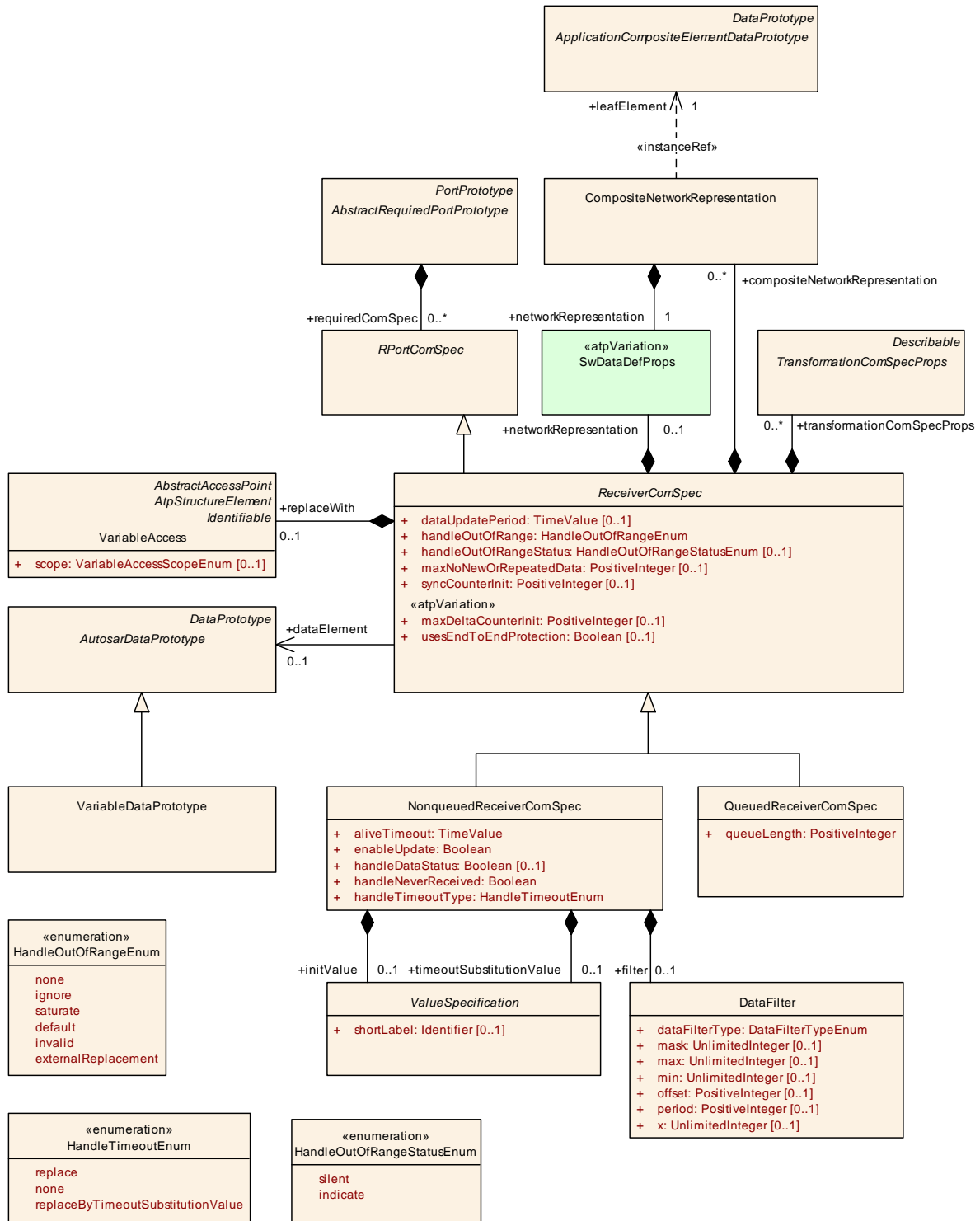


Figure 12-11: Release R4.0.1 and newer: ReceiverComSpec (hardcopy from DOC_ReceiverComSpec)

12.1.6 Error classification

The wrapper uses the standard E2E error codes of E2E library functions, which are extended with additional error codes.

[UC_E2E_0302]:

Where applicable, the following error status shall be used by E2E Wrapper functions within byte 3 of the return value, in addition to the error codes already defined by [SWS_E2E_00047] (chapter 7.1):

Type or error or status	How should the caller of E2E Wrapper handle it	Related code	Value [hex]
OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status <code>_INITIAL</code> , <code>_OK</code> , or <code>_OKSOMELOST</code> . This means that no Data has been lost since the last correct data reception.	Production	<code>E2EPW_STATUS_OK</code>	0x0
Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.	Production	<code>E2EPW_STATUS_NONEW DATA</code>	0x1
Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.	Production	<code>E2EPW_STATUS_WRONG CRC</code>	0x2
NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet	Production	<code>E2EPW_STATUS_SYNC</code>	0x3
Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.	Production	<code>E2EPW_STATUS_INITIAL</code>	0x4
Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status <code>_INITIAL</code> , <code>_OK</code> , or <code>_OKSOMELOST</code> .	Production	<code>E2EPW_STATUS_REPEATED</code>	0x8
OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter ($1 < \text{DeltaCounter} \leq \text{MaxDeltaCounter}$) with respect to the most recent Data received with Status <code>_INITIAL</code> , <code>_OK</code> , or <code>_OKSOMELOST</code> . This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.	Production	<code>E2EPW_STATUS_OKSOMELOST</code>	0x20
Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big ($\text{DeltaCounter} > \text{MaxDeltaCounter}$) with respect to the most recent Data received with Status <code>_INITIAL</code> , <code>_OK</code> , or <code>_OKSOMELOST</code> . This means that too many Data in the sequence have been probably lost since the last correct/initial reception.	Production	<code>E2EPW_STATUS_WRONG SEQUENCE</code>	0x40

Table 12-1: Error codes of E2E Wrapper functions (in addition to E2E Library error codes)

Note that the previous versions of E2E Library (R3.2.1, R4.0.1, R4.0.2) returned the value 0x10 as `E2EPW_STATUS_OK`, so in case of upgrade of E2E libraries from those versions, the SW-Cs need an update.

[UC_E2E_0303]:

Where applicable, the following error flags shall be used by E2E Wrapper functions on byte 1 of the return value, in addition to the error codes already defined by [SWS_E2E_00047] (chapter 7.1):

Type or error or status	How should the caller of E2E Wrapper handle it	Related code	Value [hex]
Extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0.	Integration or production	<code>E2EPW_E_DESERIALIZATION</code>	0x3
The control fields computed by Write1 and Write2 are not equal, i.e. status of voting between Write1 and Write2 failed	Integration or production	<code>E2EPW_E_REDUNDANCY</code>	0x5

Table 12-2: Error codes of E2E Wrapper functions (in addition to E2E Library error codes)

[SWS_E2E_00314] The caller of the E2E Wrapper functions *should* handle the errors/status defined in UC_E2E_0302 and UC_E2E_0303 according to the column “How do caller of E2E shall handle it”. (RS_E2E_08528)

In other words, the E2E library does not define any integration errors for itself, it does not call DEM nor DET. However, the caller of E2E library uses the return values of E2E functions and does the corresponding error handling.

12.1.7 E2E Protection Wrapper routines

There are two ways how the wrapper is generated. The first way is to have single channel functions Read and Write. The second way is to have redundant functions Write1, Write2, Read1 and Read2. Typically, the user should use either single channel or redundant function sets.

[UC_E2E_00293] The parameter <instance> of the E2E Protection Wrapper routines shall be present if and only if the calling software component is multiply instantiated. Because in the current release multiple instantiation of software components is not supported by E2E Protection wrapper, this means that the optional parameter <instance> shall never be present. (RS_E2E_08528)

Because the above may change in future (the support for multiple instances may be introduced), and because of the goal to have the same API as the corresponding API of RTE, the optional parameter <instance> is kept.

To support future protocol and wrapper extensions on one side and the proprietary extension on the other side, the set of return values are divided (for each byte) into AUTOSAR use and proprietary use.

[UC_E2E_00304] The return values returned by the E2E Wrapper read/write functions shall be used as follows:

- For byte 1, 2 and 3 the set of return values ranging from 0x00 to 0x7F (i.e. decimal 0 to 127) is restricted for usage within AUTOSAR specifications only and shall not be used for proprietary return values that are not part of AUTOSAR specifications.
- For byte 1, 2 and 3 the set of return values ranging from 0x80 to 0xFE (i.e. decimal 128 to 254) is not restricted and shall be used for proprietary implementation specific return values that are not part of AUTOSAR specifications.
- For byte 1, 2 and 3 the value 0xFF (i.e. decimal 255) represents the invalid value. (RS_E2E_08527)

Only a subset of return values out of the set of restricted return values (i.e. 0x00 to 0x7F) is used within AUTOSAR specifications today, the remaining ones are reserved for future use by AUTOSAR.

[UC_E2E_00328] Redundant wrapper routines shall use separate configuration and state data structures for each of the redundant channels. (RS_E2E_08527)

E.g. use `config1_<p>_<o>/state1_<p>_<o>` for channel 1 and `config2_<p>_<o>/state2_<p>_<o>` for channel 2, as indicated in the code example in 12.1.9.1.

12.1.7.1 Single channel wrapper routines and init routines

12.1.7.1.1 E2EPW_Write_<p>_<o>

[UC_E2E_00279]

Service Name	E2EPW_Write_<p>_<o>	
Syntax	<pre>uint32 E2EPW_Write_<p>_<o> (Rte_Instance <instance>, -- <data>)</pre>	
Service ID [hex]	0x1b	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Write function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (inout)	<data>	Data element to be protected and sent. The parameter is inout, because this function invokes E2E_PXXProtect function, which updates the values of control fields. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (out)	None	
Return value	uint32	<p>The byte 0 (lowest byte) is the status of Rte_Write function:</p> <ul style="list-style-type: none"> RTE_E_COM_STOPPED - the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only) RTE_E_SEG_FAULT - a segmentation violation is detected in the handed over parameters to the RTE API. No transmission is executed RTE_E_OK - data passed to communication service successfully <p>The byte 1 is the status of runtime checks done within E2E Protection Wrapper function:</p> <ul style="list-style-type: none"> E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function E2EPW_Write completed successfully <p>The byte 2 is the return value of E2E_PXXProtect function:</p> <ul style="list-style-type: none"> E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer

		<p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2E_E_OK - Function E2E_PXXProtect completed successfully</p> <p>The byte 3 is a placeholder for future use and takes the following values</p> <p>E2E_E_OK - default case</p>
Description	Initiates a safe explicit sender-receiver transmission of a safety-related data element with data semantic. It protects data with E2E Library function E2E_PXXProtect and then it calls the corresponding RTE_Write function.	
Available via	E2E.h	

](RS_E2E_08528)

[UC_E2E_00280] The function E2EPW_Write_<p>_<o>() shall:

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
2. Invoke E2E Library function E2E_PXXProtect()
3. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, store the computed CRC/Counter in the data element
4. Invoke Rte_Write_<p>_<o>()](RS_E2E_08528)

See also Figure 12-12: E2EPW_Write sequence diagram and Figure 12-17: E2EPW_Write activity diagram.

12.1.7.1.2 E2EPW_WriteInit_<p>_<o>

[UC_E2E_00300]

Service Name	E2EPW_WriteInit_<p>_<o>	
Syntax	Std_ReturnType E2EPW_WriteInit_<p>_<o> (Rte_Instance <instance>)	
Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	Status of runtime checks:

		E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function completed successfully
Description	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via	E2E.h	

](RS_E2E_08528)

[UC_E2E_00301] The function E2EPW_WriteInit_<p>_<o> shall initialize the E2E_PXXProtectStateType_<p>_<o> with the following values:
Counter = 0](RS_E2E_08528)

12.1.7.1.3 E2EPW_Read_<p>_<o>

[UC_E2E_00165]

Service Name	E2EPW_Read_<p>_<o>	
Syntax	uint32 E2EPW_Read_<p>_<o> (Rte_Instance <instance>, -- <data>)	
Service ID [hex]	0x1c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Read function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Read function.
Parameters (inout)	None	
Parameters (out)	<data>	Parameter to pass back the received data. The pointer to the OUT. parameter <data> must remain valid until the function call returns.
Return value	uint32	The byte 0 (lowest byte) is the status of Rte_Read function: RTE_E_INVALID - data element invalid RTE_E_MAX_AGE_EXCEEDED - data element outdated RTE_E_NEVER_RECEIVED - No data received since system start or partition restart RTE_E_UNCONNECTED - Indicates that the receiver port is not connected. RTE_E_OK - data read successfully

		<p>The byte 1 is the status of runtime checks done within E2E Protection Wrapper function, plus including bit extension checks:</p> <p>E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer</p> <p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0.</p> <p>E2E_E_OK - Function E2EPW_Read completed successfully</p> <p>The byte 2 is the return value of E2E_PXXCheck function:</p> <p>E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXCheck is a NULL pointer</p> <p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXCheck is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2E_PXXCheck (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2E_E_OK - Function E2E_PXXCheck completed successfully</p> <p>The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function.</p> <p>E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.</p> <p>E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.</p> <p>E2EPW_STATUS_INITIAL - Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.</p> <p>E2EPW_STATUS_REPEATED - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.</p> <p>E2EPW_STATUS_OK - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.</p> <p>E2EPW_STATUS_OKSOMELOST - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter ($1 < \text{DeltaCounter} = \text{MaxDeltaCounter}$) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range</p> <p>E2EPW_STATUS_WRONGSEQUENCE - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big ($\text{DeltaCounter} > \text{MaxDeltaCounter}$) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have</p>
--	--	---

		been probably lost since the last correct/initial reception. E2EPW_STATUS_SYNC - NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
Description	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	
Available via	E2E.h	

](RS_E2E_08528)

[UC_E2E_00192] The function E2EPW_Read_<p>_<o>() shall:

1. Invoke Rte_Read_<p>_<o>()
2. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
3. Invoke E2E Library function E2E_PXXCheck()
4. Do the deserialization check.](RS_E2E_08528)

See also Figure 12-13: E2EPW_Read sequence diagram and Figure 12-14: E2EPW_Read activity diagram.

12.1.7.1.4 E2EPW_ReadInit_<p>_<o>

[UC_E2E_00296]

Service Name	E2EPW_ReadInit_<p>_<o>	
Syntax	Std_ReturnType E2EPW_ReadInit_<p>_<o> (Rte_Instance <instance>)	
Service ID [hex]	0x16	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_Return-Type	Status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or

	postcondition) E2E_E_OK - Function completed successfully
Description	The function reinitializes the corresponding data structure after a detected error or at startup.
Available via	E2E.h

] (RS_E2E_08528)

[UC_E2E_00297] The function E2EPW_ReadInit_<p>_<o> shall initialize the E2E_PXXCheckStateType_<p>_<o> with the following values:

```

LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
NoNewOrRepeatedDataCounter = 0
SyncCounter = 0
    ] (RS_E2E_08528)
    
```

12.1.7.2 Redundant wrapper routines

12.1.7.2.1 E2EPW_Write1_<p>_<o>

[UC_E2E_00261]

Service Name	E2EPW_Write1_<p>_<o>	
Syntax	<pre> uint32 E2EPW_Write1_<p>_<o> (Rte_Instance <instance>, -- <data>) </pre>	
Service ID [hex]	0x1d	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Write function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (inout)	<data>	Data element to be protected and sent. The parameter is inout, because this function invokes E2E_PXXProtect function, which updates the values of control fields. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (out)	None	

Return value	uint32	<p>The byte 0 (lowest byte) is equal to E2E_E_OK (because Rte_Write is not invoked)</p> <p>The byte 1 is the status of runtime checks done within E2E Protection Wrapper function:</p> <p>E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer</p> <p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2E_E_OK - Function E2EPW_Write completed successfully</p> <p>The byte 2 is the return value of E2E_PXXProtect function:</p> <p>E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer</p> <p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2E_E_OK - Function E2E_PXXProtect completed successfully</p> <p>The byte 3 is a placeholder for future use and takes the following values:</p> <p>E2E_E_OK - default case</p>
Description	It protects data with E2E Library function E2E_PXXProtect. it does not call the corresponding RTE_Write function.	
Available via	E2E.h	

] (RS_E2E_08528)

[UC_E2E_00262] The function E2EPW_Write1_<p>_<o>() shall:

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
2. Invoke E2E Library function E2E_PXXProtect()
3. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, store the computed CRC/Counter in the data element.] (RS_E2E_08528)

See also Figure 12-18: E2EPW_Write1 activity diagram.

12.1.7.2.2 E2EPW_Write2_<p>_<o>

[UC_E2E_00263]

Service Name	E2EPW_Write2_<p>_<o>
Syntax	<pre>uint32 E2EPW_Write2_<p>_<o> (Rte_Instance <instance>, -- <data>)</pre>

Service ID [hex]	0x1e	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Write function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Write function.
	<data>	Data element to be protected and sent. The parameter is IN, because this function compares the calculated protection fields from E2EPW_Write1 with independently calculated fields from invoking E2E_PXXProtect. Nothing is changed in <data> in case of success. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint32	<p>The byte 0 (lowest byte) is the status of Rte_Write function: RTE_E_COM_STOPPED - the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only) RTE_E_SEG_FAULT - a segmentation violation is detected in the handed over parameters to the RTE API. No transmission is executed RTE_E_OK - data passed to communication service successfully The byte 1 is the status of runtime Protects done within E2E Protection Wrapper function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2EPW_E_REDUNDANCY - The control fields computed by Write1 and Write2 are not equal, i.e. status of voting between Write1 and Write2 failed E2E_E_OK - Function E2EPW_Write completed successfully The byte 2 is the return value of E2E_PXXProtect function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function E2E_PXXProtect completed successfully The byte 3 is a placeholder for future use and takes the following values: E2E_E_OK - default case</p>
Description	Initiates a safe explicit sender-receiver transmission of a safety-related data element with data semantic. It protects data with E2E Library function E2E_PXXProtect,	

	compares the computed control fields with the ones computed by Write1, and then it calls the corresponding RTE_Write function.
Available via	E2E.h

](RS_E2E_08528)

[UC_E2E_00264] The function E2EPW_Write2_<p>_<o>() shall:

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
2. Invoke E2E Library function E2E_PXXProtect()
3. Execute voting on control fields between Write1 and Write2
4. Invoke Rte_Write_<p>_<o>() .](RS_E2E_08528)

See also Figure 12-19: E2EPW_Write2 activity diagram.

12.1.7.2.3 E2EPW_WriteInit1_<p>_<o>

[SWS_E2E_00318]

Service Name	E2EPW_WriteInit1_<p>_<o>	
Syntax	uint8 E2EPW_WriteInit1_<p>_<o> (Rte_Instance <instance>)	
Service ID [hex]	0x17	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	The byte 0 is the status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function completed successfully
Description	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via	E2E.h	

](RS_E2E_08528)

[SWS_E2E_00322] The function E2EPW_WriteInit1_<p>_<o> shall initialize the E2E_PXXProtectStateType_<p>_<o> related to redundant channel 1 with the following values:

Counter = 0. (RS_E2E_08528)

12.1.7.2.4 E2EPW_WriteInit2_<p>_<o>

[SWS_E2E_00319]

Service Name	E2EPW_WriteInit2_<p>_<o>	
Syntax	uint8 E2EPW_WriteInit2_<p>_<o> (Rte_Instance <instance>)	
Service ID [hex]	0x18	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	The byte 0 is the status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function completed successfully
Description	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via	E2E.h	

(RS_E2E_08528)

[SWS_E2E_00323] The function E2EPW_WriteInit2_<p>_<o> shall initialize the E2E_PXXProtectStateType_<p>_<o> related to redundant channel 2 with the following values:

Counter = 0. (RS_E2E_08528)

12.1.7.2.5 E2EPW_Read1_<p>_<o>

[UC_E2E_00265]

Service Name	E2EPW_Read1_<p>_<o>	
Syntax	<pre>uint32 E2EPW_Read1_<p>_<o> (Rte_Instance <instance>, -- <data>)</pre>	
Service ID [hex]	0x1f	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Read function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Read function.
Parameters (inout)	None	
Parameters (out)	<data>	Parameter to pass back the received data. The pointer to the OUT. parameter <data> must remain valid until the function call returns.
Return value	uint32	<p>The byte 0 (lowest byte) is the status of Rte_Read function:</p> <ul style="list-style-type: none"> RTE_E_INVALID - data element invalid RTE_E_MAX_AGE_EXCEEDED - data element outdated RTE_E_NEVER_RECEIVED - No data received since system start or partition restart RTE_E_UNCONNECTED - Indicates that the receiver port is not connected. RTE_E_OK - data read successfully <p>The byte 1 is the status of runtime checks done within E2E Protection Wrapper function:</p> <ul style="list-style-type: none"> E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0. E2E_E_OK - Function E2EPW_Read completed successfully <p>The byte 2 is the return value of E2E_PXXCheck function:</p> <ul style="list-style-type: none"> E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXCheck is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXCheck is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2E_PXXCheck (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function E2E_PXXCheck completed successfully <p>The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX,</p>

	<p>determined by the Check function.</p> <p>E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.</p> <p>E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.</p> <p>E2EPW_STATUS_INITIAL - Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.</p> <p>E2EPW_STATUS_REPEATED - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.</p> <p>E2EPW_STATUS_OK - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.</p> <p>E2EPW_STATUS_OKSOMELOST - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.</p> <p>E2EPW_STATUS_WRONGSEQUENCE - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception</p> <p>E2EPW_STATUS_SYNC - NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.</p>
Description	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.
Available via	E2E.h

](RS_E2E_08528)

[UC_E2E_00266] The function E2EPW_Read1_<p>_<o>() shall:

1. Invoke Rte_Read_<p>_<o>()
2. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
3. Invoke E2E Library function E2E_PXXCheck()
4. Do the deserialization check.](RS_E2E_08528)

See also Figure 12-15: E2EPW_Read1 activity diagram.

12.1.7.2.6 E2EPW_Read2_<p>_<o>

[UC_E2E_00267]

Service Name	E2EPW_Read2_<p>_<o>	
Syntax	<pre>uint32 E2EPW_Read2_<p>_<o> (Rte_Instance <instance>, -- <data>)</pre>	
Service ID [hex]	0x20	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Read function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Read function.
	<data>	The received data to be checked. The parameter is IN, because this function re-performs the checks on the already received data (by E2EPW_Read1_<p>_<o>). Nothing is changed in <data>. The pointer to the IN parameter <data> must remain valid until the function call returns.
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint32	<p>The byte 0 (lowest byte) equal to RTE_E_OK (because Rte_Read is not invoked)</p> <p>The byte 1 is the status of runtime checks done within E2E Protection Wrapper function:</p> <p>E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer</p> <p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0.</p> <p>E2E_E_OK - Function E2EPW_Read completed successfully</p> <p>The byte 2 is the return value of E2E_PXXCheck function:</p> <p>E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXCheck is a NULL pointer</p>

	<p>E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXCheck is erroneous, e.g. out of range</p> <p>E2E_E_INTERR - An internal error has occurred in E2E_PXXCheck (e.g. error detected by program flow monitoring, violated invariant or postcondition)</p> <p>E2E_E_OK - Function E2E_PXXCheck completed successfully The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function.</p> <p>E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.</p> <p>E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.</p> <p>E2EPW_STATUS_INITIAL - Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.</p> <p>E2EPW_STATUS_REPEATED - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.</p> <p>E2EPW_STATUS_OK - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.</p> <p>E2EPW_STATUS_OKSOMELOST - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter ($1 < \text{DeltaCounter} = \text{MaxDeltaCounter}$) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.</p> <p>E2EPW_STATUS_WRONGSEQUENCE - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big ($\text{DeltaCounter} > \text{MaxDeltaCounter}$) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception</p> <p>E2EPW_STATUS_SYNC - NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.</p>
Description	The function re-checks the data received with corresponding function Read1 by means of execution of E2E_PXXCheck.
Available via	E2E.h

](RS_E2E_08528)

[UC_E2E_00268] The function E2EPW_Read2_<p>_<o>() shall:

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
2. Invoke E2E Library function E2E_PXXCheck()
3. Do the deserialization check.](RS_E2E_08528)

See also Figure 12-16: E2EPW_Read2 activity diagram.

12.1.7.2.7 E2EPW_ReadInit1_<p>_<o>

[SWS_E2E_00320][

Service Name	E2EPW_ReadInit1_<p>_<o>	
Syntax	uint8 E2EPW_ReadInit1_<p>_<o> (Rte_Instance <instance>)	
Service ID [hex]	0x19	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	The byte 0 is the status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function completed successfully
Description	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via	E2E.h	

](RS_E2E_08528)

[SWS_E2E_00324] The function E2EPW_ReadInit1_<p>_<o> shall initialize the E2E_PXXCheckStateType_<p>_<o> related to redundant channel 1 with the following values:

```
LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
```

NoNewOrRepeatedDataCounter = 0
SyncCounter = 0.](RS_E2E_08528)

12.1.7.2.8 E2EPW_ReadInit2_<p>_<o>

[SWS_E2E_00321]

Service Name	E2EPW_ReadInit2_<p>_<o>	
Syntax	uint8 E2EPW_ReadInit2_<p>_<o> (Rte_Instance <instance>)	
Service ID [hex]	0x1a	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint8	The byte 0 is the status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function completed successfully
Description	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via	E2E.h	

](RS_E2E_08528)

[SWS_E2E_00325] The function E2EPW_ReadInit2_<p>_<o> shall initialize the E2E_PXXCheckStateType_<p>_<o> related to redundant channel 2 with the following values:

LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
NoNewOrRepeatedDataCounter = 0
SyncCounter = 0.](RS_E2E_08528)

12.1.8 E2EPW Routines Diagrams

12.1.8.1 Sequence Diagrams – Read and Write

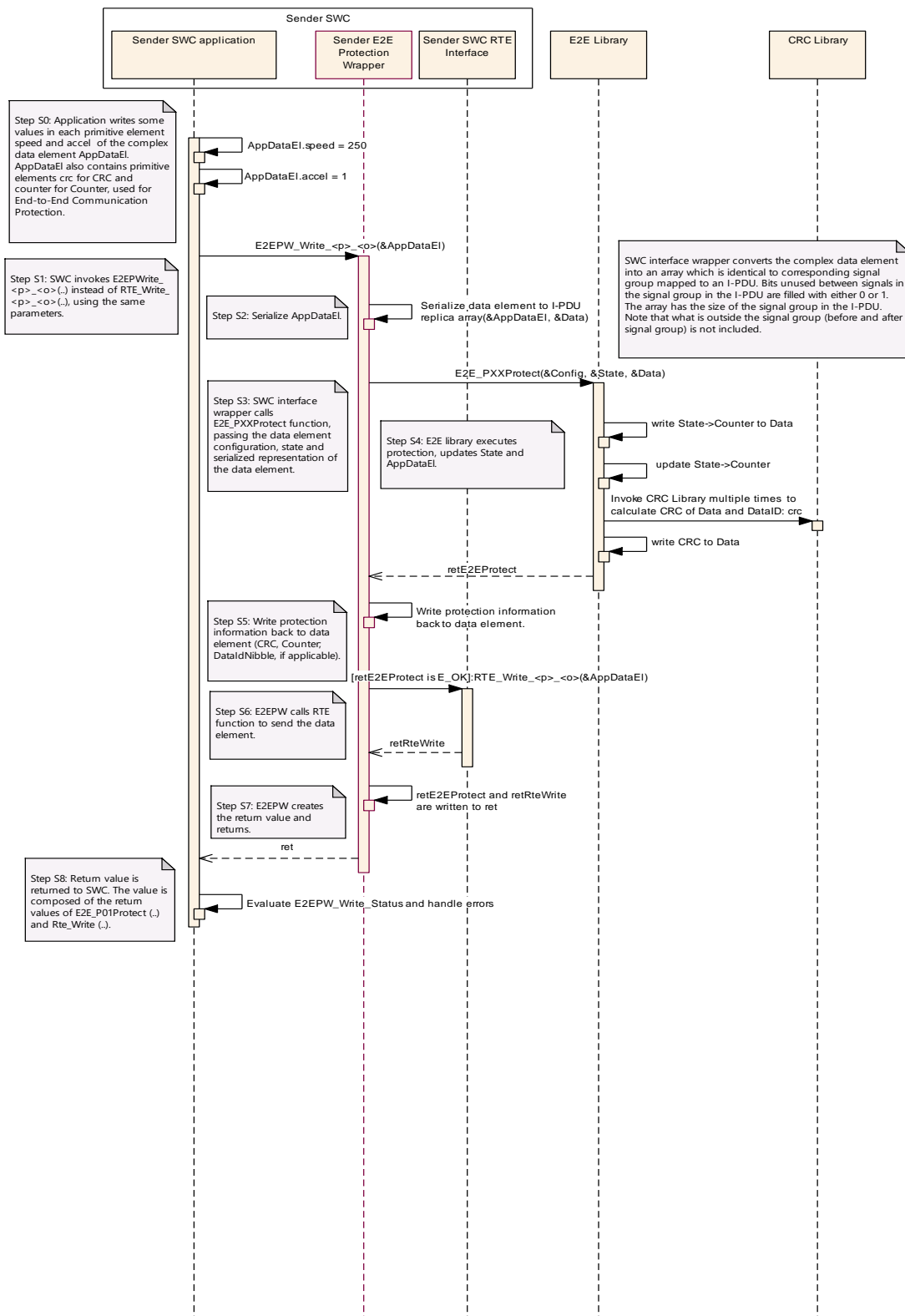


Figure 12-12: E2EPW_Write sequence diagram

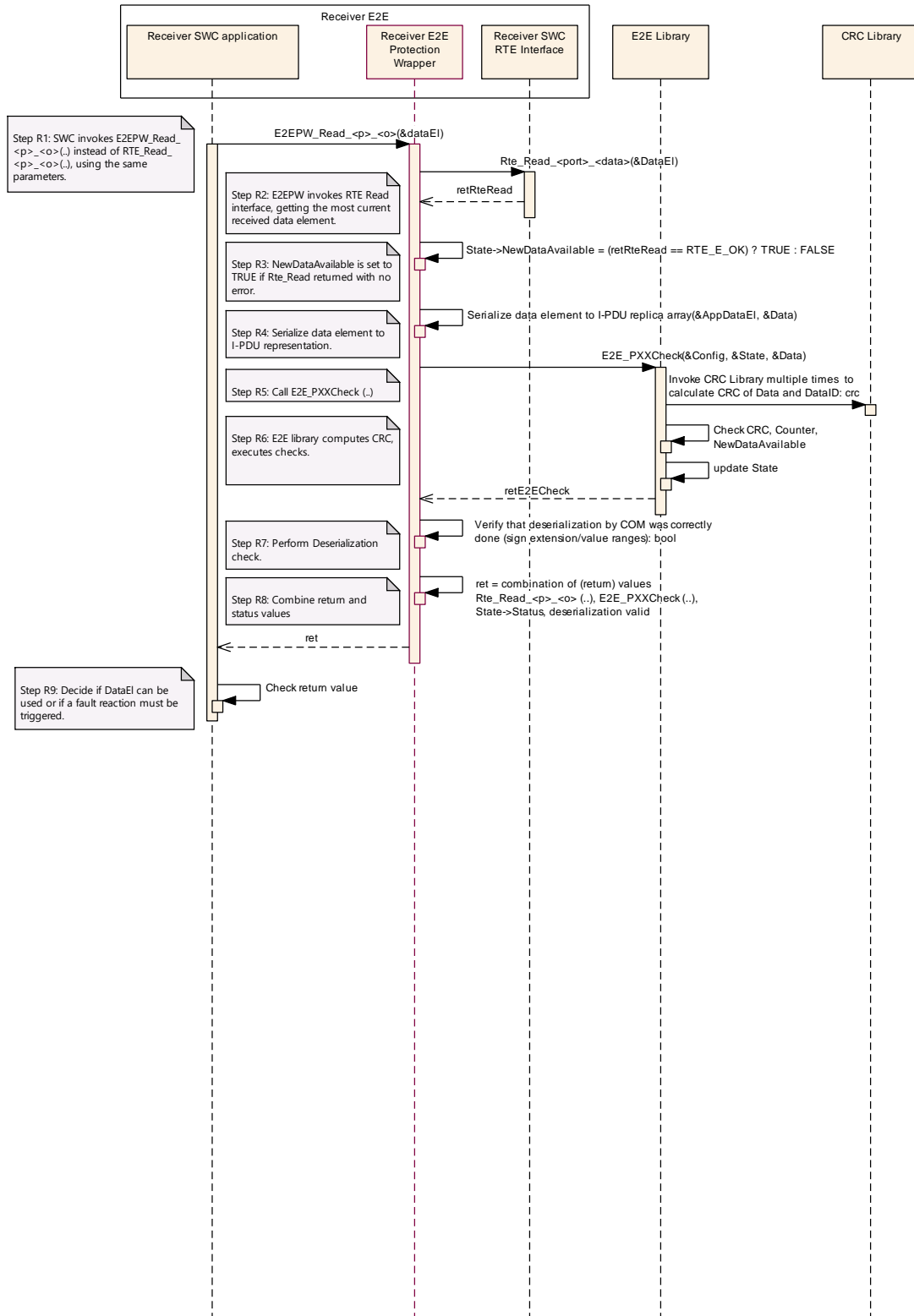


Figure 12-13: E2EPW_Read sequence diagram

12.1.8.2 Activity Diagrams – E2EPW Read, Read1 and Read2

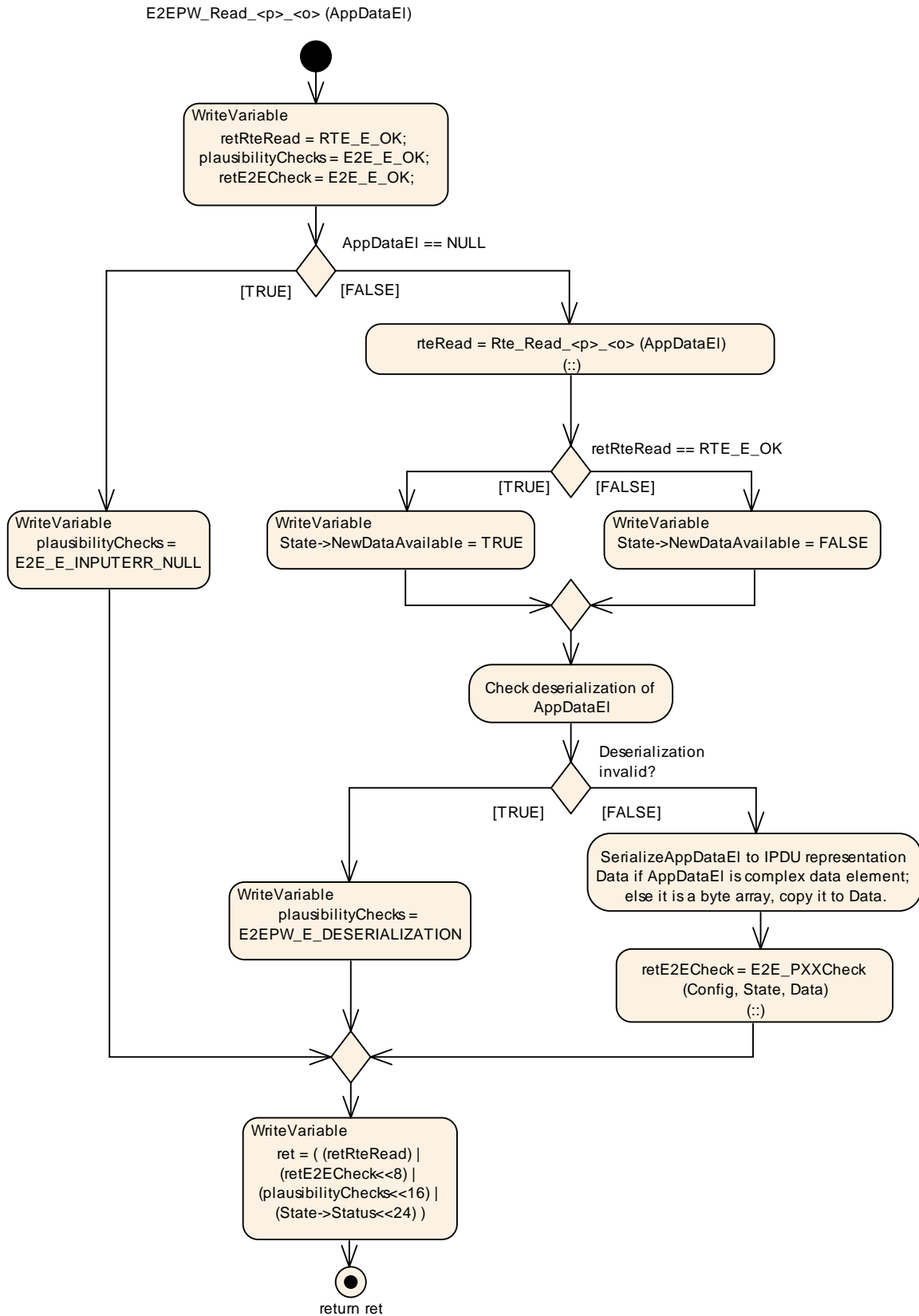


Figure 12-14: E2EPW_Read activity diagram

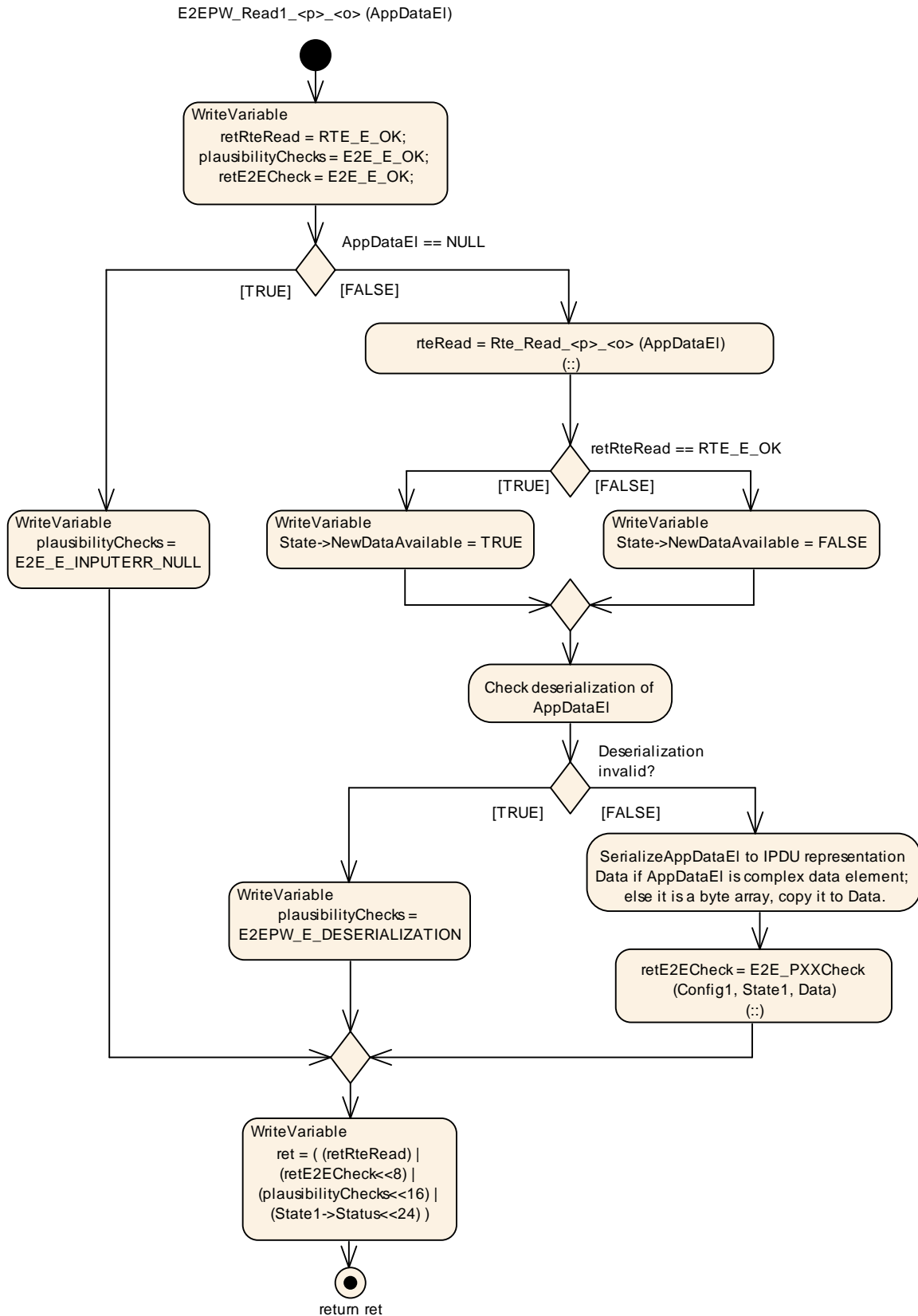


Figure 12-15: E2EPW_Read1 activity diagram

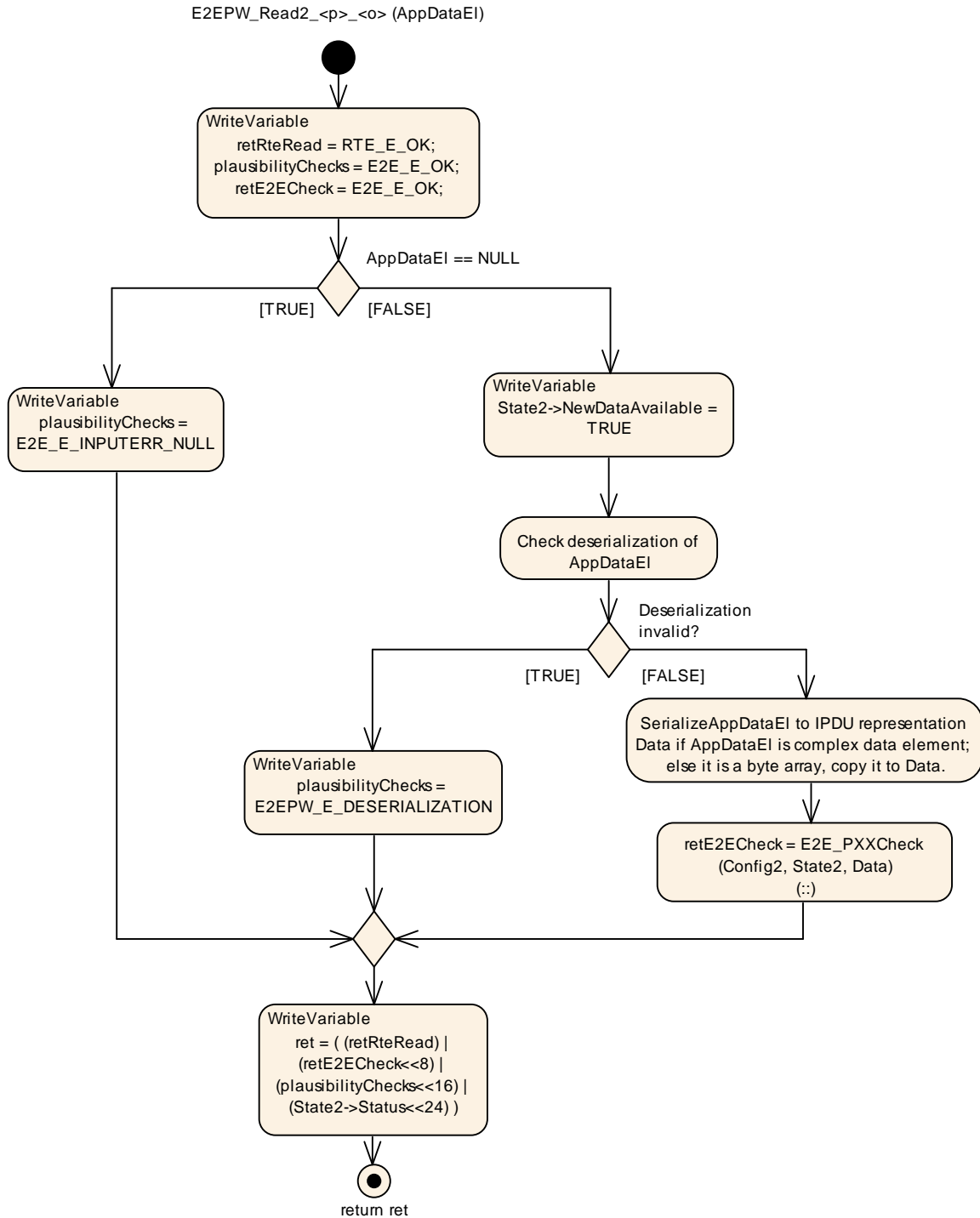


Figure 12-16: E2EPW_Read2 activity diagram

12.1.8.3 Activity Diagrams – E2EPW Write, Write1 and Write2

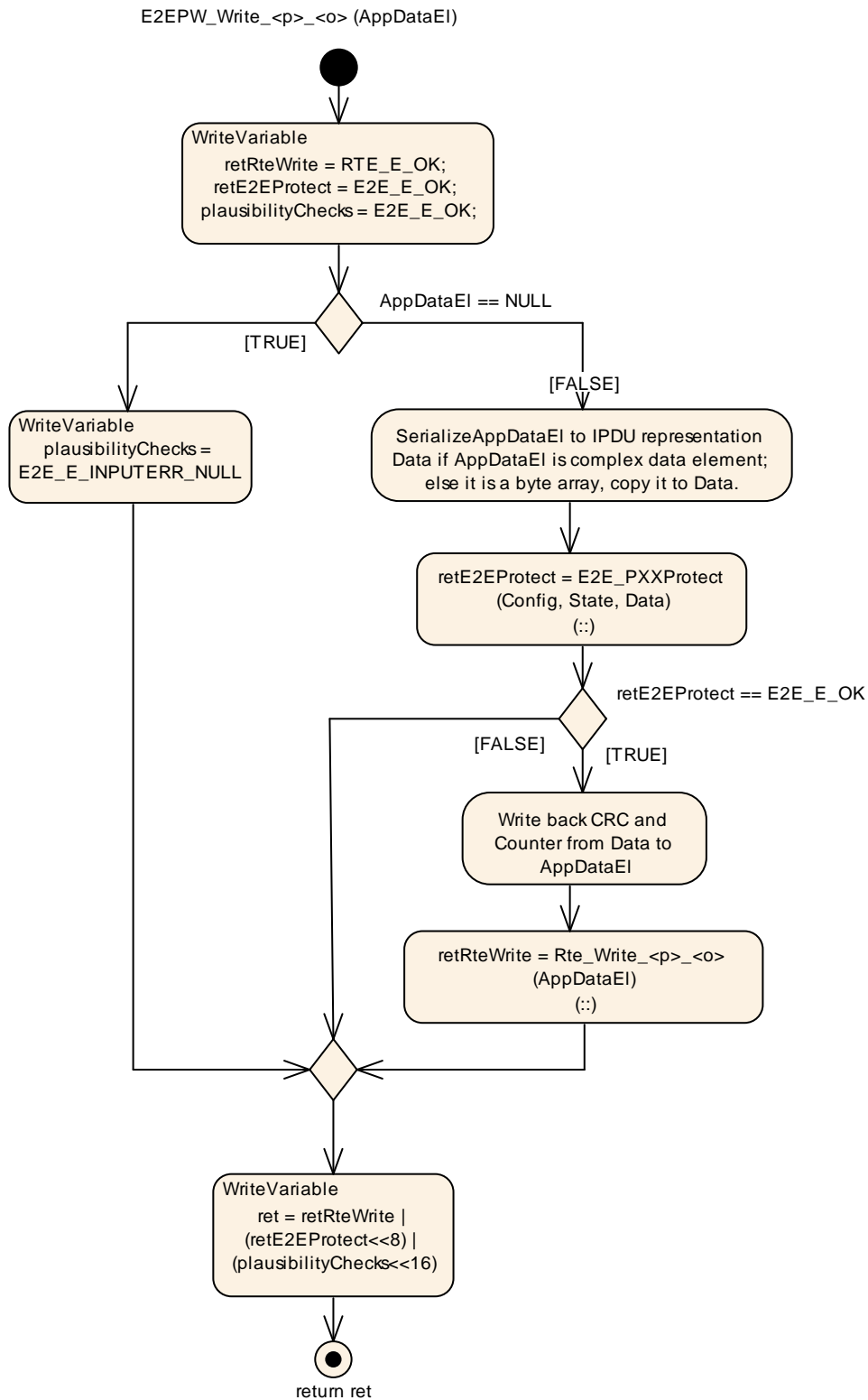


Figure 12-17: E2EPW_Write activity diagram

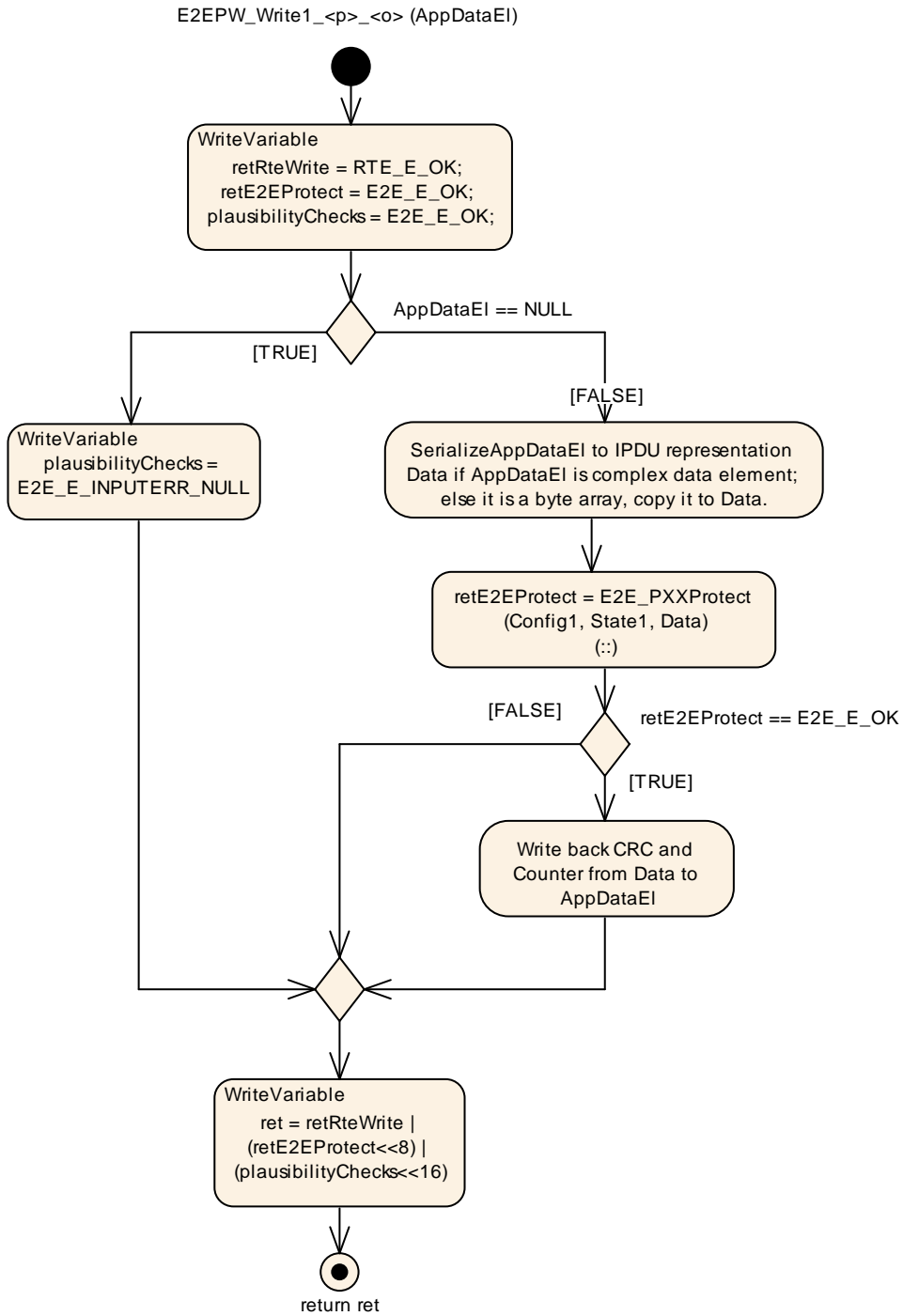


Figure 12-18: E2EPW_Write1 activity diagram

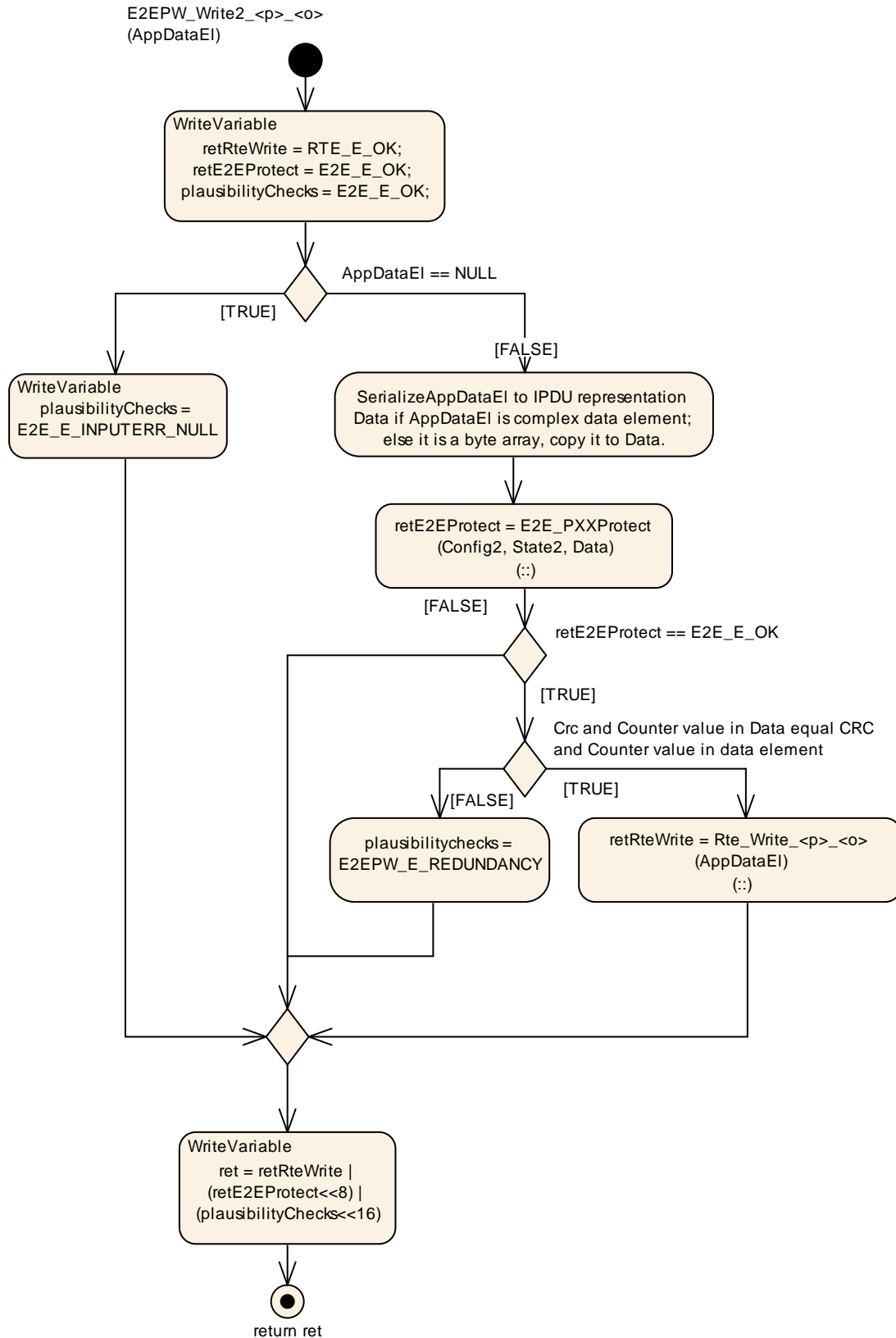


Figure 12-19: E2EPW_Write2 activity diagram

12.1.9 Code Example

Important:

To enable proper memory mapping by the AUTOSAR memmap methodology and to enable the use of init functions, function-static and function-constant variables cannot be used and must be defined on module level. To avoid name clashes, they shall be suffixed.

The suffixes used shall be:

1. For functions E2EPW_Write_<p>_<o> and E2EPW_Read_<p>_<o>: with suffix “_<p>_<o>” (e.g. variable_<p>_<o> instead of variable)
2. For functions E2EPW_Write1_<p>_<o> and E2EPW_Read1_<p>_<o>: with suffix “1_<p>_<o>” (e.g. variable_<p>_<o> instead of variable)
3. For functions E2EPW_Write2_<p>_<o> and E2EPW_Read2_<p>_<o>: with suffix “2_<p>_<o>” (e.g. variable_<p>_<o> instead of variable)

In the code example, the suffix is formatted like this: _<p>_<o>

This is to emphasize that <p> and <o> are placeholders.

The below code example illustrates the possible implementation of E2E Protection wrapper. The example shows Profile 1, but this is applicable also for Profile 2.

Note: The below code is only pseudocode to provide a better understanding of the intention of the functionality and does not claim to be correct or to be a reference implementation.

The code example shows the single channel and redundant wrapper. The single channel wrapper is the simplest way to keep the application logic of SW-C independent from data protection, where the wrapper to protect the data on behalf of the application.

The redundant wrapper requires that it is invoked twice by application, but it has the following additional features:

1. Code redundancy:
 - a. For each Rte_Write* function, there are corresponding E2EPW_Write1* and E2EPW_Write2* functions
 - b. For each Rte_Read* function, there are corresponding E2EPW_Read1* and E2EPW_Read2* functions
2. Time diversity:
 - a. The functions E2EPW_Write1* and E2EPW_Write2* on the sender side and E2EPW_Read1* and E2EPW_Read2* are executed one after each other.
3. Data redundancy:
 - a. All data used by the redundant wrapper, apart from application data element, is redundant
 - b. The application data element is instantiated by Rte one time only. To mitigate faults, is written/read by application at each call of E2EPW_Write1, E2EPW_Write2, E2EPW_Read1, E2EPW_Read2.

There are no configuration options in AUTOSAR templates to select which wrapper shall be generated. Either redundant or single channel functions should be generated

(generating both single channel and redundant wrapper calls for the same SW-Cs would signify generation of dead code). The choice which wrapper is generated may be a global option in the wrapper generator. Alternatively, a wrapper may be able to generate either single-channel or redundant wrapper only.

Write/Read symmetry

On the sender side, the two functions Write1 and Write2 compute (create) the values for the control fields (which are CRC and counter for Profiles 1 and 2). Because two different outputs (one from Write1 and one from Write2) are generated, they are compared by Write2 before sending them through RTE.

On the receiver side however, there is no creation of control fields. Instead, they are double-checked (once by Read1 and once by Read2). Therefore, it is checked if both Read1 and Read2 functions agree on the check results (e.g. if both Read1 and Read2 report that the CRC is correct). This voting is done by comparing byte 2 of return values of Read1 and Read2 (and is executed by application (no by the wrapper).

12.1.9.1 Code Example – Sender SW-C

12.1.9.1.1 Sender– E2EPW_WriteInit, E2EPW_WriteInit1 and E2EPW_WriteInit2

This chapter presents an example implementation of functions

E2EPW_WriteInit_<p>_<o>(), E2EPW_WriteInit1_<p>_<o>() and E2EPW_WriteInit2_<p>_<o>() as well as definition of the module-static configuration and state data structures. $\langle DataLength / 8 \rangle$ is the dataLength configuration value divided by 8 (to represent the length in bytes). The example configuration values are random, but valid values.

```
static const E2E_P01ConfigType Config_<p>_<o> =
{ 8,      /* CounterOffset */
  0,      /* CRCOffset */
  0x12,   /* DataID */
  12,     /* DataIDNibbleOffset */
  E2E_P01_DATAID_BOTH, /* DataIDMode */
  64,     /* DataLength */
  1,      /* MaxDeltaCounterInit */
  3,      /* MaxNoNewOrRepeatedData */
  2,      /* SyncCounterInit */
};

static E2E_P01ProtectStateType State_<p>_<o> =
{ 0 /* Counter */
};

/* byte array for call of E2Elib */
static uint8 Data_<p>_<o>[<math>\langle DataLength / 8 \rangle</math>];

Std_ReturnType E2EPW_WriteInit_<p>_<o>(Rte_Instance Instance) {
  State_<p>_<o>.Counter = 0;
  return E2E_E_OK;
}
```


For redundant wrapper:

```

static const E2E_P01ConfigType Config1_<p>_<o> =
{ 8,      /* CounterOffset */
  0,      /* CRCOffset */
  0x12,   /* DataID */
  12,     /* DataIDNibbleOffset */
  E2E_P01_DATAID_BOTH, /* DataIDMode */
  64,     /* DataLength */
  1,      /* MaxDeltaCounterInit */
  3,      /* MaxNoNewOrRepeatedData */
  2,      /* SyncCounterInit */
};

static E2E_P01ProtectStateType State1_<p>_<o> =
{ 0 /* Counter */
};

static const E2E_P01ConfigType Config2_<p>_<o> =
{ 8,      /* CounterOffset */
  0,      /* CRCOffset */
  0x12,   /* DataID */
  12,     /* DataIDNibbleOffset */
  E2E_P01_DATAID_BOTH, /* DataIDMode */
  64,     /* DataLength */
  1,      /* MaxDeltaCounterInit */
  3,      /* MaxNoNewOrRepeatedData */
  2,      /* SyncCounterInit */
};

static E2E_P01ProtectStateType State2_<p>_<o> =
{ 0 /* Counter */
};

/* byte array for call of E2Elib - only one is needed for redundant
wrapper */
static uint8 Data_<p>_<o>[<DataLength * 8>];

Std_ReturnType E2EPW_WriteInit1_<p>_<o>(Rte_Instance Instance) {
  State1_<p>_<o>.Counter = 0;
return E2E_E_OK;
}

Std_ReturnType E2EPW_WriteInit2_<p>_<o>(Rte_Instance Instance) {
  State2_<p>_<o>.Counter = 0;
return E2E_E_OK;
}

```

12.1.9.1.2 Sender –E2EPW_Write and E2EPW_Write1

This chapter presents an example implementation of functions `E2EPW_Write_<p>_<o>()` and `E2EPW_Write1_<p>_<o>()`.

12.1.9.1.2.1 Generation / Initialization

Generation/Initialization: RTE generates a complex data element (case A) or an opaque uint8 array (Case B).

Case A (complex data type):

The RTE Generator generates the complex data element. The complex data element has additional two data elements `crc` and `counter`, which are unused by SW-C application part, but only by the E2E Protection Wrapper.

```
typedef struct {
    uint8 crc; /* additional data el, unused by SW-C */
    uint8 counter; /* additional data el, unused by SW-C */
    uint8 dataIDHighByteNibble; /* for nibble configuration of
                                E2E profile 1 only */
    uint16 speed; /* 16-bit, but 12 bits used in I-PDU*/
    uint8 accel; /* 8-bit number, 4 bits used */
} DataType;
...
static DataType AppDataElVal;
static DataType *AppDataEl = &AppDataElVal;
```

Case B (array):

The RTE Generator generates an opaque uint8 array.

```
static uint8 AppDataEl[8];
```

12.1.9.1.2.2 Step S0

Step S0: Application writes the values in a complex data type:

Case A (complex data type)

```
AppDataEl->speed = U16_V_MAX; /*16-bit number, 12 bits used */
AppDataEl->accel = U8_G_EARTH; /* 8-bit number, 4 bits used */
```

Case B (array):

```
AppDataEl [1] = (U8_G_EARTH & 0x0F) << 4;
AppDataEl [2] = (uint8) (U16_V_MAX & 0x00FF);
AppDataEl [3] = (uint8) (U16_V_MAX) >> 8;
AppDataEl [3] |= 0xF0;
AppDataEl [4] = 0xFF;
```

12.1.9.1.2.3 Step S1

Step S1: Application calls E2E Protection Wrapper.

```
/* single channel - Write */
uint32 wrapperRet = E2EPW_Write_<p>_<o>(Instance, AppDataE1);
```

The redundant step is identical, apart from “1” suffix:

```
/* redundant - Write1 */
uint32 wrapperRet1 = E2EPW_Write1_<p>_<o>(Instance, AppDataE1);
```

12.1.9.1.2.4 Step S2

Step S2: The E2E Wrapper (E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()) checks for wrong parameters from SW-C and it creates a data copy:

Case A (complex data type):

The E2E Protection Wrapper (E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()) serializes the data to the layout identical with the layout of the corresponding signal group in the I-PDU. It fills in unused bits with a predefined pattern, e.g. ‘1’-s (as defined in unusedBitPattern of ISignalPdu; To get ‘1’-s, unusedBitPattern is 0xFF).

Note that there can be several signal groups in an I-PDU, each protected or not with E2E by means of the wrapper. This means that the Data_<p>_<o> array contains the representation of only one signal group mapped to the I-PDU.

```
Std_ReturnType plausibilityChecks = E2E_E_OK;
...
/* example of possible plausibility checks */
if (AppDataE1 == NULL) {
    return (E2E_E_INPUTERR_NULL << 8);
}

/* Data has the same layout as serialized signal group in I-PDU.
   Initialize all bytes of Data[] with the unused bit pattern (called
   unusedBitPattern in system template. */

Data_<p>_<o>[0] = 0;

/* in accel, only 4 bits are used, they go
   To high nibble of Data[1], next to Counter. */
Data_<p>_<o>[1] = (AppDataE1->accel & 0x0F) << 4;

/* in speed, only 8+4 bits are used.
   low byte of speed goes to Data[2].*/
Data_<p>_<o>[2] = (AppDataE1->speed & 0x00FF);

/* low nibble of high byte goes to Data[3] */
Data_<p>_<o>[3] = (AppDataE1->speed & 0x0F00) >> 8;

/* high nibble of high byte of Data[3] is unused, so it is set with 1s
   on each unused bit */
Data_<p>_<o>[3] |= 0xF0;

/* Data[4] is unused but transmitted, so it is explicitly set
```

```

        to 0xFF*/
Data_<p>_<o>[4] = 0xFF;
    
```

The above example is illustrated by the figure below:

```

typedef struct {
    Uint8 crc; /* additional data el, unused by SW-C */
    Uint8 counter; /* additional data el, unused by SW-C */
    Uint16 speed; /* 16-bit, but 12 bits used in I-PDU*/
    Uint8 accel; /* 16-bit, but 12 bits used in I-PDU*/
} DataEl;
    
```

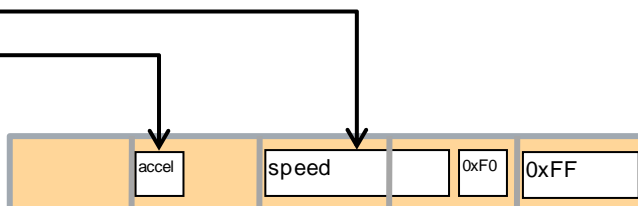


Figure 12-20: Mapping of Data elements into I-PDU

Case B (array):

The E2E Protection Wrapper (`E2EPW_Write_<p>_<o>`, `E2EPW_Write1_<p>_<o>()`) simply casts the data element to the array and copies it:

```

Std_ReturnType plausibilityChecks = E2E_E_OK;
...
/* example of possible plausibility checks */
if (AppDataEl == NULL) {
    return (E2E_E_INPUTERR_NULL<< 8);
}

memcpy(Data_<p>_<o>, AppDataEl, 8);
    
```

12.1.9.1.2.5 Step S3

Step S3: E2E Protection Wrapper (`E2EPW_Write_<p>_<o>`, `E2EPW_Write1_<p>_<o>()`) calls the E2E library to protect the data element.

```

/* single channel - Write */
Std_ReturnType retE2EProtect = E2E_P01Protect(&Config_<p>_<o>,
&State_<p>_<o>, Data_<p>_<o>);
    
```

The redundant step is identical, apart from “1” suffix:

```

/* redundant - Write1 */
Std_ReturnType retE2EProtect = E2E_P01Protect(&Config1_<p>_<o>,
&State1_<p>_<o>, Data_<p>_<o>);
    
```

12.1.9.1.2.6 Step S4

Step S5: E2E executes protection, updates State and AppDataEl.

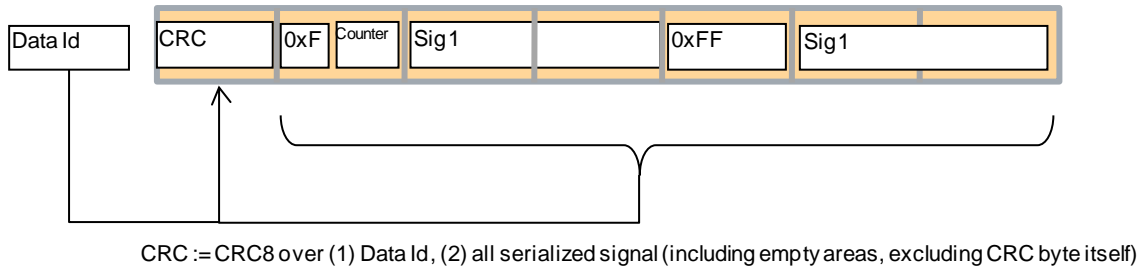


Figure 12-21: Step 4

12.1.9.1.2.7 Step S5

Step S5: The E2E Protection Wrapper (`E2EPW_Write_<p>_<o>`, `E2EPW_Write1_<p>_<o>()`) copies back the control fields to `AppDataEl`.

Case A (complex data type):

```
AppDataEl->crc = Data_<p>_<o>[0]; /* Copy CRC from byte 0 */
AppDataEl->counter = Data_<p>_<o>[1]&0x0F; /* Copy counter from byte 1 */
*/
```

This is illustrated by the Figure 12-22:

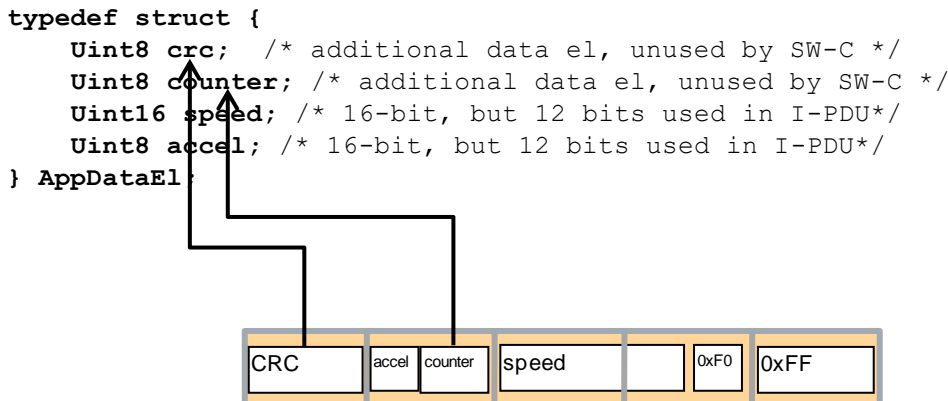


Figure 12-22: Copy back of CRC and alive from I-PDU copy to data element

Case B (array):

```
AppDataEl[0] = Data_<p>_<o>[0]; /* Copy CRC from byte 0 */
AppDataEl[1] = (AppDataEl[1]&0xF0) | (Data_<p>_<o>[1]&0x0F); /* Copy CRC */
```

12.1.9.1.2.8 Step S6

Step S6: Single channel Wrapper (`E2EPW_Write_<p>_<o>`) calls RTE function to send the data element and returns the extended status to SW-C.

```
/* Single channel - Write*/
Std_ReturnType retRteWrite = Rte_Write_<p>_<o>(Instance, AppDataEl);
```

Redundant wrapper (`E2EPW_Write2_<p>_<o>()`) in step S7 does *not* call `Rte_Write_<p>_<o>()` function.

```
/* Redundant - Write1 */
Std_ReturnType retRteWrite = E2E_E_OK;
```

12.1.9.1.2.9 Step S7

Step S7: The E2E Wrapper creates the return value and returns.

```
return ((retRteWrite) | (retE2EProtect<<16)
 | (plausibilityChecks<<8));
```

12.1.9.1.2.10 Step S8

Step S8: Caller SW-C checks the return value of the wrapper and handles errors, if any. This behavior is specific to the application.

```
/* single channel - Write */
if(wrapperRet != 0) swc_error_handler(wrapperRet);
```

```
/* redundant - Write1 */
if(wrapperRet1 != 0) swc_error_handler(wrapperRet1);
```

12.1.9.1.3 Sender - E2EPW_Write2

This chapter presents an example implementation of function `E2EPW_Write2_<p>_<o>()`.

12.1.9.1.3.1 Step S10

Step S10: Application writes the values in a complex data type.

Step S10-S19 are only for the redundant scenario. The step S10 is just the repetition of S0 on the same values. The application rewrites the data in `AppDataE1`. The values must be identical to the values written in step S0, otherwise the voting in step S17 will fail. This redundant write is to prevent some faults related to `AppDataE1` (e.g. corruption from outside, random memory fault on that area)

12.1.9.1.3.2 Step S11

Steps S11-S18 represent the steps of the function `E2EPW_Write2_<p>_<o>()`.

Step S11: Application calls E2E Protection Wrapper for the second time, this time `E2EPW_Write2_<p>_<o>()` function.

```
uint32 wrapperRet2 = E2EPW_Write2_<p>_<o>(Instance, AppDataE1);
```

12.1.9.1.3.3 Step S12

The step S13 (of function `E2EPW_Write2_<p>_<o>()`) is 100% identical to Step S2 (of function `E2EPW_Write1_<p>_<o>()`).

12.1.9.1.3.4 Step S13

Step S3: E2E Protection Wrapper (`E2EPW_Write2_<p>_<o>()`) calls the E2E library to protect the data element.

```
/* redundant - Write2 */
Std_ReturnType retE2EProtect = E2E_P01Protect(Config2_<p>_<o>,
State2_<p>_<o>, Data_<p>_<o>);
```

12.1.9.1.3.5 Step S14

The step S14 (of function `E2EPW_Write2_<p>_<o>()`) is 100% identical to Step S4 (of function `E2EPW_Write1_<p>_<o>()`).

12.1.9.1.3.6 Step S15–skipped

Contrary to Step S5, there is no copying back of control fields back to `AppDataE1` in `E2EPW_Write2_<p>_<o>()`.

12.1.9.1.3.7 Steps S16

At this stage, the Wrapper (`E2EPW_Write2_<p>_<o>()`) has to its disposition the following:

1. `AppDataE1` containing data partly from Step S0 and Step S10:
 - a. application data filled in by the SW-C in Step S10
 - b. `crc` and `counter` filled in by `E2EPW_Write1_<p>_<o>()` based on `AppDataE1` filled in in step S0.
2. `Data` containing:
 - a. `crc` and `counter` filled in by `E2EPW_Write2_<p>_<o>()`, based on `AppDataE1` from Step S10.

There are two safety mechanisms provided:

1. The control fields (`crc` and `counter` from `AppDataE1` and from `Data`) are binary compared by the voter. By this means, the results `Write1` and `Write2` are voted by the sender
2. The `AppDataE1` at this stage contains the application data filled in step S10, but the control fields are computed on data filled in Step S0. In case of error (difference) that has not been detected by the sender voter, the receiver serves as the second voter.

Only in case of successful voting, the data (application data from second round and control fields from first round) is transmitted through RTE.

Case A (structure):

```
if( (AppDataE1->counter != (Data_port1_del[1] & 0x0F)) ||
(AppDataE1->crc != (Data_port1_del[0]          ) ) )
```

```

    plausibilityChecks = E2EPW_E_REDUNDANCY; /* 0x05 */

Std_ReturnType retRteWrite = E2E_E_OK;

/* Write data regardless if redundancy error detected ... */
retRteWrite = Rte_Write_<p>_<o>(Instance, AppDataE1);

```

Case B (array):

```

if( ((AppDataE1[1] & 0x0F) != (Data_port1_de1[1] & 0x0F)) ||
    (AppDataE1[0] != (Data_port1_de1[0] ) ) )
    plausibilityChecks= E2EPW_E_REDUNDANCY; /* 0x05 */

Std_ReturnType retRteWrite = E2E_E_OK;

/* Write data regardless if redundancy error detected ... */
retRteWrite = Rte_Write_<p>_<o>(Instance, AppDataE1);

```

12.1.9.1.3.8 Step S17

Step S17: The E2E Wrapper creates the return value and returns.

```

return ((retRteWrite) | (retE2EProtect<< 16)
        | (plausibilityChecks<< 8));

```

12.1.9.1.3.9 Step S18

Step S18: Caller SW-C checks the return value (of function `E2EPW_Write2_<p>_<o>()`) and handles errors, if any. It also compares the return values of `E2EPW_Write2_<p>_<o>()` against return value of `E2EPW_Write1_<p>_<o>()`.

```

if(wrapperRet2 != 0) swc_error_handler(wrapperRet2);

```

12.1.9.2 Code Example – Receiver SW-C

12.1.9.2.1 Receiver - E2EPW_ReadInit, E2EPW_ReadInit1 and E2EPW_ReadInit2

This chapter presents an example implementation of functions `E2EPW_ReadInit_<p>_<o>()`, `E2EPW_ReadInit1_<p>_<o>()` and `E2EPW_ReadInit2_<p>_<o>()` as well as definition of the module-static configuration and state data structures. $\langle DataLength / 8 \rangle$ is the dataLength configuration value divided by 8 (to represent the length in bytes). The example configuration values are random, but valid values.

```

static const E2E_P01ConfigType Config_<p>_<o> =
{ 8,      /* CounterOffset */
  0,      /* CRCOffset */
  0x12,   /* DataID */
  12,    /* DataIDNibbleOffset */
  E2E_P01_DATAID_BOTH, /* DataIDMode */
  64,    /* DataLength */

```



```

1, /* MaxDeltaCounterInit */
    3, /* MaxNoNewOrRepeatedData */
    2, /* SyncCounterInit */
};

static E2E_P01CheckStateType State_<p>_<o> =
{ 0, /* LastValidCounter */
0, /* MaxDeltaCounter */
TRUE, /* WaitForFirstData */
FALSE, /* NewDataAvailable */
0, /* LostData */
    E2E_P01STATUS_NONEWDATA, /* Status */
0, /* SyncCounter */
    0 /* NoNewOrRepeatedDataCounter */
};

/* byte array for call of E2Elib */
static uint8 Data_<p>_<o>[<DataLength / 8>];

Std_ReturnType E2EPW_ReadInit_<p>_<o>(Rte_Instance Instance) {
    State_<p>_<o>.LastValidCounter = 0;
    State_<p>_<o>.MaxDeltaCounter = 0;
    State_<p>_<o>.WaitForFirstData = TRUE;
    State_<p>_<o>.NewDataAvailable = FALSE;
    State_<p>_<o>.LostData = 0;
    State_<p>_<o>.Status = E2E_P01STATUS_NONEWDATA;
    State_<p>_<o>.SyncCounter = 0;
    State_<p>_<o>.NoNewOrRepeatedDataCounter = 0;
    return E2E_E_OK;
}
    
```

For redundant wrapper:

```

static const E2E_P01ConfigType Config1_<p>_<o> =
{ 8, /* CounterOffset */
    0, /* CRCOffset */
    0x12, /* DataID */
    12, /* DataIDNibbleOffset */
    E2E_P01_DATAID_BOTH, /* DataIDMode */
    64, /* DataLength */
1, /* MaxDeltaCounterInit */
    3, /* MaxNoNewOrRepeatedData */
    2, /* SyncCounterInit */
};

static const E2E_P01ConfigType Config2_<p>_<o> =
{ 8, /* CounterOffset */
    0, /* CRCOffset */
    0x12, /* DataID */
    12, /* DataIDNibbleOffset */
    E2E_P01_DATAID_BOTH, /* DataIDMode */
    64, /* DataLength */
1, /* MaxDeltaCounterInit */
    3, /* MaxNoNewOrRepeatedData */
    2, /* SyncCounterInit */
};
    
```

```

static E2E_P01CheckStateType State1_<p>_<o> =
{ 0, /* LastValidCounter */
0, /* MaxDeltaCounter */
TRUE, /* WaitForFirstData */
FALSE, /* NewDataAvailable */
0, /* LostData */
E2E_P01STATUS_NONEWDATA, /* Status */
0, /* SyncCounter */
0 /* NoNewOrRepeatedDataCounter */
};

static E2E_P01CheckStateType State2_<p>_<o> =
{ 0, /* LastValidCounter */
0, /* MaxDeltaCounter */
TRUE, /* WaitForFirstData */
FALSE, /* NewDataAvailable */
0, /* LostData */
E2E_P01STATUS_NONEWDATA, /* Status */
0, /* SyncCounter */
0 /* NoNewOrRepeatedDataCounter */
};

/* byte array for call of E2Elib */
static uint8 Data_<p>_<o>[<DataLength * 8>];

Std_ReturnType E2EPW_ReadInit1_<p>_<o>(Rte_Instance Instance) {
    State1_<p>_<o>.LastValidCounter = 0;
    State1_<p>_<o>.MaxDeltaCounter = 0;
    State1_<p>_<o>.WaitForFirstData = TRUE;
    State1_<p>_<o>.NewDataAvailable = FALSE;
    State1_<p>_<o>.LostData = 0;
    State1_<p>_<o>.Status = E2E_P01STATUS_NONEWDATA;
    State1_<p>_<o>.SyncCounter = 0;
    State1_<p>_<o>.NoNewOrRepeatedDataCounter = 0;
    return E2E_E_OK;
}

Std_ReturnType E2EPW_ReadInit2_<p>_<o>(Rte_Instance Instance) {
    State2_<p>_<o>.LastValidCounter = 0;
    State2_<p>_<o>.MaxDeltaCounter = 0;
    State2_<p>_<o>.WaitForFirstData = TRUE;
    State2_<p>_<o>.NewDataAvailable = FALSE;
    State2_<p>_<o>.LostData = 0;
    State2_<p>_<o>.Status = E2E_P01STATUS_NONEWDATA;
    State2_<p>_<o>.SyncCounter = 0;
    State2_<p>_<o>.NoNewOrRepeatedDataCounter = 0;
    return E2E_E_OK;
}

```

12.1.9.2.2 Receiver - E2EPW_Read and E2EPW_Read1

This chapter presents an example implementation of functions `E2EPW_Read_<p>_<o>()` and `E2EPW_Read1_<p>_<o>()`.

12.1.9.2.2.1 Generation / Initialization

Generation/Initialization: RTE generates a complex data element (case A) or an opaque uint8 array (Case B).

Case A (complex data type):

The RTE Generator generates the complex data element for the receiver. The complex data element has additional two data elements crc and counter, which are unused by SW-C application part, but only by the E2E Protection Wrapper. The data element is the same on the sender and on the receiver SW-C.

```
typedef struct {
    uint8 crc;          /* additional data el, unused by SW-C */
    uint8 counter;     /* additional data el, unused by SW-C */
    uint8 dataIDHighByteNibble; /* for nibble configuration of
                                E2E profile 1 only */
    uint16 speed;      /* 16-bit, but 12 bits used in I-PDU*/
    uint8 accel;       /* 16-bit, but 12 bits used in I-PDU*/
} DataType;

...
static DataType AppDataElVal;
static DataType *AppDataEl = &AppDataElVal;
```

Case B (array):

The RTE Generator generates an opaque uint8 array.

```
static uint8 AppDataEl[8];
```

12.1.9.2.2.2 Step R1

Step R1: Application calls E2E Protection Wrapper to get the data.

```
/* single channel - Read */
uint32 wrapperRet = E2EPW_Read_<p>_<o>(Instance, AppDataEl);
```

```
/* redundant - Read1 */
uint32 wrapperRet1 = E2EPW_Read1_<p>_<o>(Instance, AppDataEl);
```

12.1.9.2.2.3 Step R2

Step R2: Wrapper (E2EPW_Read_<p>_<o>, E2EPW_Read1_<p>_<o>()) checks the parameters and then calls RTE functionRte_Read to receive the data element.

```
Std_ReturnType plausibilityChecks = E2E_E_OK, retRteRead;
...
/* example of possible plausibility checks */
if (AppDataEl == NULL) {
    return (E2E_E_INPUTERR_NULL);
}

retRteRead = Rte_Read_<p>_<o>(Instance, AppDataEl);
```

12.1.9.2.2.4 Step R3

Step R3: NewDataAvailable is set if Rte_Read_<p>_<o>() returned without error.

```

/* single channel */
State_<p>_<o>.NewDataAvailable = (retRteRead == RTE_E_OK) ? TRUE :
FALSE;

```

Redundant wrapper:

```

/* redundant */
State1_<p>_<o>.NewDataAvailable = (retRteRead == RTE_E_OK) ? TRUE :
FALSE;

```

12.1.9.2.2.5 Step R4

Step R4:the E2E Protection Wrapper serializes the data to the layout identical with the one of the corresponding I-PDU. The E2E Protection wrapper needs to do the serialization (I-PDU from the received data), so that E2E Library can compute and check the CRC.

Case A (complex data type):

```

/* For storing the same layout as the one of I-PDU */
Data_<p>_<o>[0] = 0;

/* in accel, only 4 bits are used,
they go To high nibble of Data[1], next to Counter. */
Data_<p>_<o>[1] = (AppDataE1->accel & 0x0F) << 4;

/* in speed, only 8+4 bits are used.
low byte of speed goes to Data[2].*/
Data_<p>_<o>[2] = (AppDataE1->speed & 0x00FF);

/* low nibble of high byte goes to Data[3] */
Data_<p>_<o>[3] = (AppDataE1->speed & 0x0F00) >> 8;

/* high nibble of high byte of Data[3] is unused, so it is set with 1s
on each unused bit */
Data_<p>_<o>[3] |= 0xF0;

/* Data[4] is unused but transmitted, so it is explicitly set
to 0xFF*/
Data_<p>_<o>[4] = 0xFF;

```

Case B:

The E2E Protection Wrapper (E2EPW_Read_<p>_<o>, E2EPW_Read1_<p>_<o>()) simply casts the data element to the array and copies it:

```

/* Copy from AppDataE1 to Data */
memcpy(Data_<p>_<o>, AppDataE1, 8);

```

12.1.9.2.2.6 Step R5

Step R5: E2E Protection Wrapper calls the E2E library to check the data element.

```
/* single channel - Read */
```

```
Std_ReturnType retE2ECheck = E2E_P01Check(&Config_<p>_<o>,
&State_<p>_<o>, Data_<p>_<o>);
```

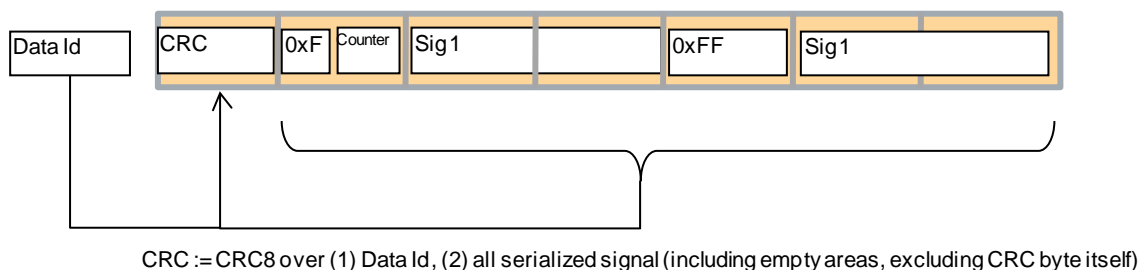
The redundant step is identical, apart from "1" suffix:

```
/* redundant - Read1 */
```

```
Std_ReturnType retE2ECheck = E2E_P01Check(&Config1_<p>_<o>,
&State1_<p>_<o>, Data_<p>_<o>);
```

Step R6

Step R6: E2E computes CRC, and executes the checks.



12.1.9.2.2.7 Step R7

Step R7: the E2E Protection Wrapper checks if the deserialization is done correctly

Case A (complex data type):

The E2E Protection Wrapper verifies that the bit extensions done by COM are done correctly. This step is needed, because unused most significant bits of primitive data elements are simply cut out (not placed in I-PDUs). On the receiver side, these unused bits shall have a specified value (e.g. they shall be 0 for unsigned numbers). Note that the unused most significant bits of signals are not related to unused bits between signals in I-PDUs.

```
/* in accel, only 4 bits are used, they go
   To high nibble of Data[1], next to Counter.
*/
```

```
if( (AppDataE1->accel & 0xF0) != 0)
plausibilityChecks = E2EPW_E_DESERIALIZATION;
```

```
/* in speed, only 8+4 bits are used.
   Topmost 4 bits shall be 0 */
```

```
if( (AppDataE1->accel & 0xF000) != 0)
plausibilityChecks = E2EPW_E_DESERIALIZATION;
```

Case B (array):

Not present, as there is no bit extension done by COM

```
plausibilityChecks = E2E_E_OK;
```

12.1.9.2.2.8 Step R8

Step R8: The E2E wrapper returns to the application.

```
/* single channel */
return ( (retRteRead) | (retE2ECheck<<16) |
         (plausibilityChecks<<8) | (uint32) (State_<p>_<o>.Status)<<24 ) ;
```

The redundant step is identical, apart from “1” suffix:

```
/* redundant */
return ( (retRteRead) | (retE2ECheck<<16) |
         (plausibilityChecks<<8) | (uint32) (State1_<p>_<o>.Status)<<24 ) ;
```

12.1.9.2.2.9 Step R9

Step R9: Caller SW-C checks the return value and handles errors, if any. This behavior is specific to the application. Then it copies the data from `AppDataE1` to application buffer and consumes it.

Note that the caller may accept some errors on byte 3 (e.g. it may accept if byte 3 equals to `E2E_PXXSTATUS_OKSOMELOST`).

Case A (complex data type):

```
/* single channel */
if ( ((wrapperRet) &0xFF != 0) ||
      ((wrapperRet>>8) &0xFF != 0) ||
      ((wrapperRet>>16) &0xFF != 0) ||
      (((wrapperRet>>24) &0xFF != E2EPW_STATUS_OKSOMELOST) &&
       ((wrapperRet>>24) &0xFF != E2EPW_STATUS_OK))
) {
    swc_error_handler(wrapperRet);
}

targetSpeed = AppDataE1->speed;
targetAccel = AppDataE1->accel;
```

```
/* redundant */
if ( ((wrapperRet1) &0xFF != 0) ||
      ((wrapperRet1>>8) &0xFF != 0) ||
      ((wrapperRet1>>16) &0xFF != 0) ||
      (((wrapperRet1>>24) &0xFF != E2EPW_STATUS_OKSOMELOST) &&
       ((wrapperRet1>>24) &0xFF != E2EPW_STATUS_OK))
) {
    swc_error_handler(wrapperRet1);
}

targetSpeed1 = AppDataE1->speed;
targetAccel1 = AppDataE1->accel;
```

Case B (array):

```
/* single channel */
if ( ((wrapperRet) &0xFF != 0) ||
```

```

        ((wrapperRet>>8 )&0xFF != 0) ||
        ((wrapperRet>>16)&0xFF != 0) ||
        (((wrapperRet>>24)&0xFF != E2EPW_STATUS_OKSOMELOST) &&
         ((wrapperRet>>24)&0xFF != E2EPW_STATUS_OK))
    ) {
    swc_error_handler(wrapperRet);
    }

    uint16 targetSpeed = (AppDataE1[2]) | (AppDataE1[3]<<8&0x0F);
    uint8 targetAccel = AppDataE1[1] >> 4;
    
```

```

    /* redundant */
    if( ((wrapperRet1 )&0xFF != 0) ||
        ((wrapperRet1>>8 )&0xFF != 0) ||
        ((wrapperRet1>>16)&0xFF != 0) ||
        (((wrapperRet1>>24)&0xFF != E2EPW_STATUS_OKSOMELOST) &&
         ((wrapperRet1>>24)&0xFF != E2EPW_STATUS_OK))
    ) {
    swc_error_handler(wrapperRet1);
    }

    if(wrapperRet1 != 0) swc_error_handler(wrapperRet1);
    uint16 targetSpeed1 = (AppDataE1[2]) | (AppDataE1[3]<<8&0x0F);
    uint8 targetAccel1 = AppDataE1[1] >> 4;
    
```

12.1.9.2.3 Receiver - E2EPW_Read2

This chapter presents an example implementation of function `E2EPW_Read2_<p>_<o>()`.

12.1.9.2.3.1 Step R10 – skipped

Value unused to numbering consistency.

12.1.9.2.3.2 Step R11

Step R11: Application calls the wrapper again.

```

uint32 wrapperRet2 = E2EPW_Read2_<p>_<o>(Instance, AppDataE1);
    
```

12.1.9.2.3.3 Step R12 – partially skipped

Contrary to step R2 RTE is not read. Both read steps use the same data from RTE. There is only checking for parameters:

```

Std_ReturnType plausibilityChecks = E2E_E_OK, retRteRead = E2E_E_OK;
...
/* example of possible plausibility checks */
if (AppDataE1 == NULL) {
    return (E2E_E_INPUTERR_NULL);
}
    
```

12.1.9.2.3.4 Steps R13

Step R13: contrary to R3, NewDataAvailable is always set.

```
/* set always to true, because Rte_Read is not invoked. */
State2_<p>_<o>.NewDataAvailable = TRUE;
```

12.1.9.2.3.5 Steps R14

The step R14 (of function E2EPW_Read2_<p>_<o>()) is 100% identical to Step R4 (of function E2EPW_Read1_<p>_<o>()).

12.1.9.2.3.6 Step R15

Step R15: E2E Protection Wrapper calls the E2E library to check the data element.

```
Std_ReturnType retE2ECheck = E2E_P01Check(Config2_<p>_<o>,
State2_<p>_<o>, Data_<p>_<o>);
```

12.1.9.2.3.7 Step R16

The step R16 (of function E2EPW_Read2_<p>_<o>()) is 100% identical to Step R6 (of function E2EPW_Read1_<p>_<o>()).

12.1.9.2.3.8 Step R17

The step R17 (of function E2EPW_Read2_<p>_<o>()) are 100% identical to Step R7 (of function E2EPW_Read1_<p>_<o>()).

12.1.9.2.3.9 Step R18

Step R8: The E2E wrapper returns to the application.

```
return ( (retRteRead) | (retE2ECheck<<16) |
(plausibilityChecks<<8) | (uint32)(State2_<p>_<o>.Status)<<24 );
```

12.1.9.2.3.10 Step R19

Step R19: Application reads the values from the complex data type, compares them (from Read1 and from Read2) and consumes them.

Case A (complex data type):

```
/* copy values from data element */
uint16 targetSpeed2 = AppDataE1->speed;
uint8 targetAccel2 = AppDataE1->accel;

/* check if E2EPW_Read2 was successful */
if(wrapperRet2 != 0) swc_error_handler(wrapperRet2);

/* Check if both Read1 and Read2 report the same status.
In particular, byte2 of ret1 and ret2 shall be identical. If not,
then it means that there is a disagreement on evaluation
```



```
    of data between Read1 and Read2 */
if(wrapperRet2 != wrapperRet1) swc_error_handlerR(wrapperRet1,
wrapperRet2);

/* check for corruption of AppDataE1 after CRC has been checked */
if(targetSpeed2 != targetSpeed1) swc_error_handlerR(wrapperRet1,
wrapperRet2);
if(targetAccel2 != targetAccel1) swc_error_handlerR(wrapperRet1,
wrapperRet2);

/* consume targetSpeed1/targetSpeed2 and targetAccel1/targetAccel2*/
```

Case B (array):

```
/* copy values from data element */
uint16 targetSpeed2 = (AppDataE1[2]) | (AppDataE1[3]<<8 & 0x0F);
uint8 targetAccel2 = AppDataE1[1] >> 4;

/* check if E2EPW_Read2 was successful */
if(wrapperRet2 != 0) swc_error_handler(wrapperRet2);

/* Check if both Read1 and Read2 report the same status.
   In particular, byte2 of ret1 and ret2 shall be identical. If not,
   then it means that there is a disagreement on evaluation
   of data between Read1 and Read2 */
if(wrapperRet2 != wrapperRet1) swc_error_handler(wrapperRet1,
wrapperRet2);

/* check for corruption of AppDataE1 after CRC has been checked */
if(targetSpeed2 != targetSpeed1) swc_error_handlerR(wrapperRet1,
wrapperRet2);
if(targetAccel2 != targetAccel1) swc_error_handlerR(wrapperRet1,
wrapperRet2);

/* consume targetSpeed1/targetSpeed2 and targetAccel1/targetAccel2*/
```

12.2COM E2E Callouts

In this approach, the E2E communication protection protects the data exchange between COM modules. The protection is done at the level of COM's signal groups, which are protected and checked by E2E Library.

This solution works with all communication models, multiplicities offered by RTE for inter-ECU communication.

The callout invokes the E2E Library, once for each E2E-protected signal group in a given I-PDU.

This solution can be used in the systems where the integrity of operation of COM and RTE is provided.

12.2.1 Functional overview

For each I-PDU, there is a separate callout function. Each I-PDU callout function “knows” if and how each signal group of the I-PDU needs to be protected/checked. This means that the callout invokes the E2E Library functions with appropriate settings and state parameters. The E2E Library does now “know” signal groups and their settings – entire information is passed as function parameters to E2E library functions.

On both receiver and sender side, if a callout returns TRUE, then COM continues. If a COM E2E Callout returns FALSE, then COM stops to process the given I-PDU (in this cycle). The COM E2E Callout returns FALSE if and only if there is an internal error, e.g. program flow error, data corruption error in E2E Lib.

The sender callout always TRUE if there are no runtime errors detected (e.g. wrong parameter), otherwise FALSE. The receiver callout receiver returns TRUE if there are no runtime errors detected and the result of the check is either E2E_P02STATUS_OK or E2E_P02STATUS_OKSOMELOST.

The diagram below summarizes the COM E2E Callout solution on the sender side. The SW-C is completely not impacted, and only additional activities in COM is invocation of the generated callout (step 6). If the return value from the callout is TRUE, then the IpduData modified by E2E Library is then transmitted by PDU router. If false, then COM stops further processing of this I-PDU in this cycle.

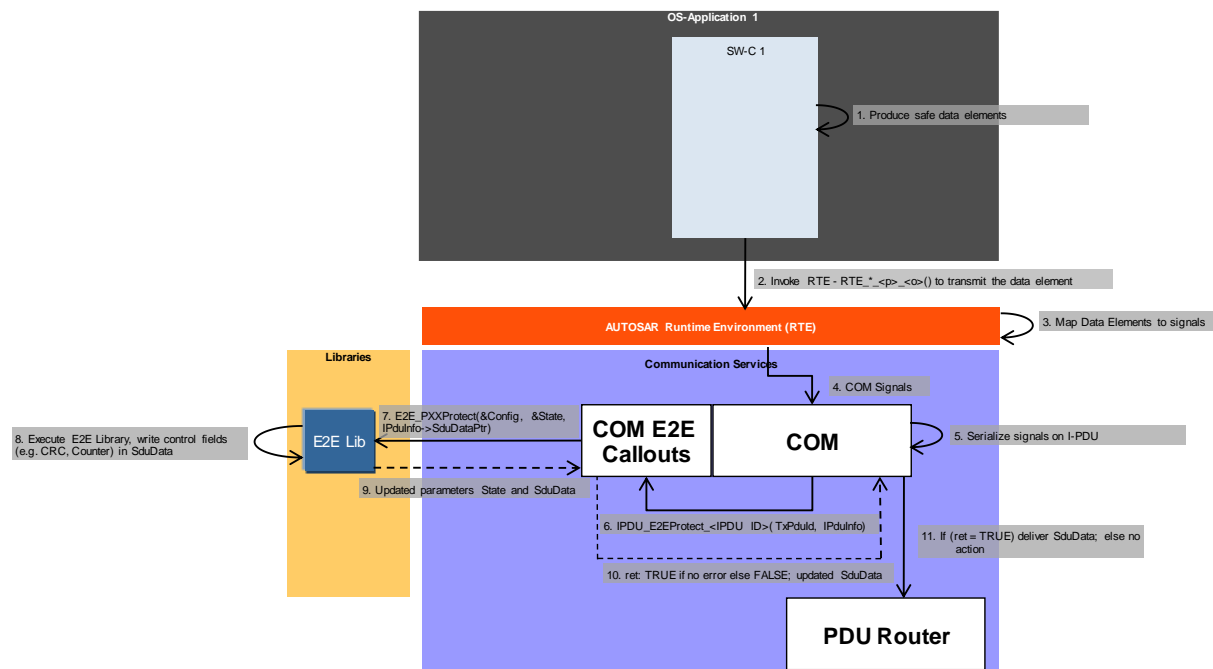


Figure 12-23: Callout – overall flow – P-port

The diagram below summarizes the COM E2E Callout solution. The very important step is that the E2E Library overwrites CRC byte in the signal group by the check status bits

(E2E_PXXCheckStateType). Then, this overwritten CRC byte is converted by COM to signals and then by RTE to data elements. As a result, the SW-C receives in the CRC data element the E2E check bits, and not the CRC value.

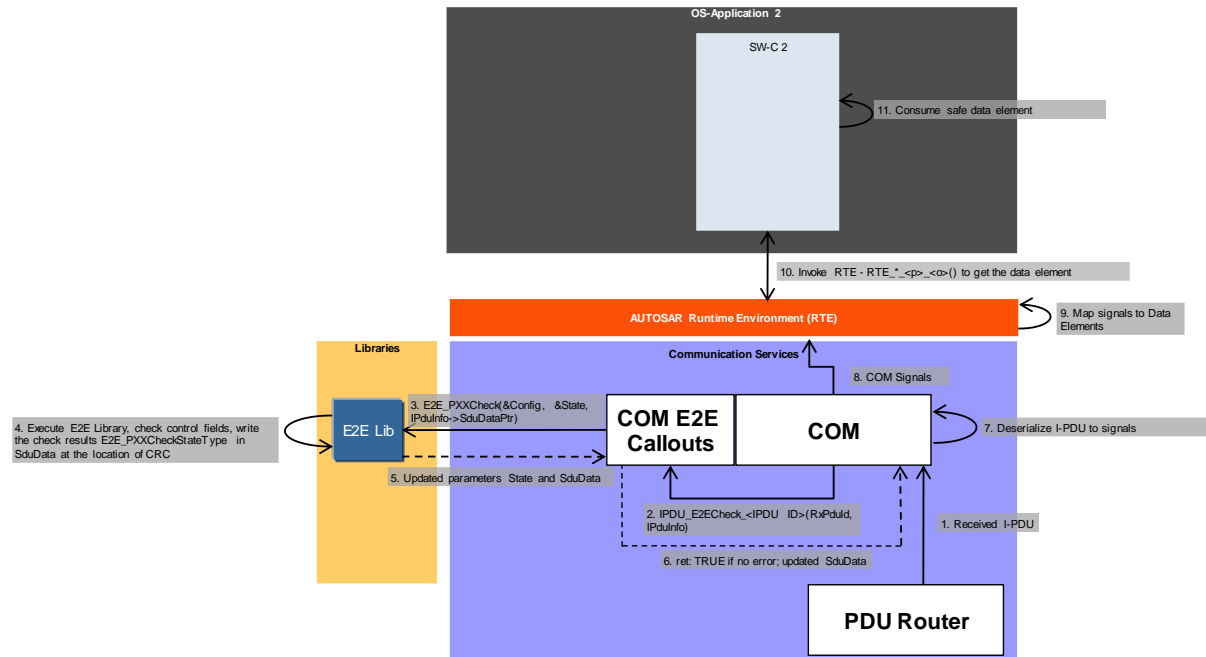


Figure 12-24: Callout – overall flow – R-port

Sending/Calling

On the sender COM side, when the I-PDU has been built from signals and the conversions (e.g. Endianness) have taken place, and the I-PDU is ready, then COM calls a callout function. There is a separate callout for each I-PDU (if defined). Once the callout returns, COM invokes the PDU Router to transmit the data (function PduR_ComTransmit).

The callout function is generated to protect the signal groups of one I-PDU and simply invokes the E2E Library (once per each E2E-protected signal group) with the correct hard-coded settings. The hard-coded settings have been generated from the settings described in the previous section.

When the callout returns TRUE, COM invokes PduR_ComTransmit(), to route the I-PDU through the network.

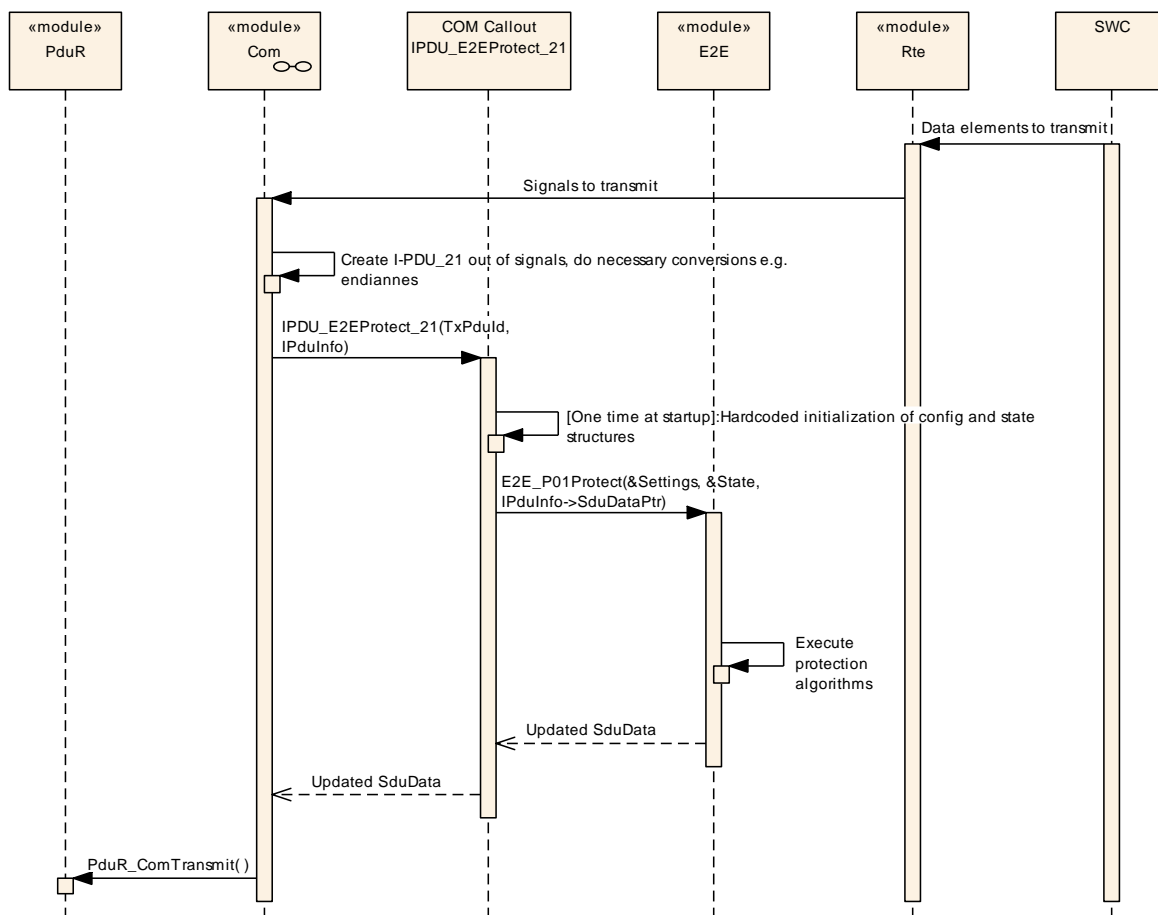


Figure 12-25: Callout – sequence – sending

According to COM SWS, the callouts shall conform to the following syntax:
 boolean <IPDU_CalloutName> (PduIdType TxPduId, const PduInfoType* PduInfoPtr)

[UC_E2E_00250] The transmission callout for usage with E2E shall be the following:
 IPDU_E2EProtect_<IPDU ID>(PduIdType TxPduId, PduInfoType* PduInfoPtr).

For example, the callout to protect the I-PDU with handle 21 shall have the name
 IPDU_E2EProtect_21(). (RS_E2E_08528)

Reception

On the receiver COM side, when the I-PDU is available at PDU Router, PDU Router invokes COM's function COM_RxIndication(). COM then calls the generated I-PDU callout (if configured for the given I-PDU). The callout, generated specifically for that I-PDU, calls the E2E Library with specific parameters (once for each E2E-protected signal group). The E2E Library executes the checks and stores the check results in the status. Once E2E Library check function returns, the callout copies the status into the CRC byte, so that it can be analyzed, if needed, by receiver SW-C.

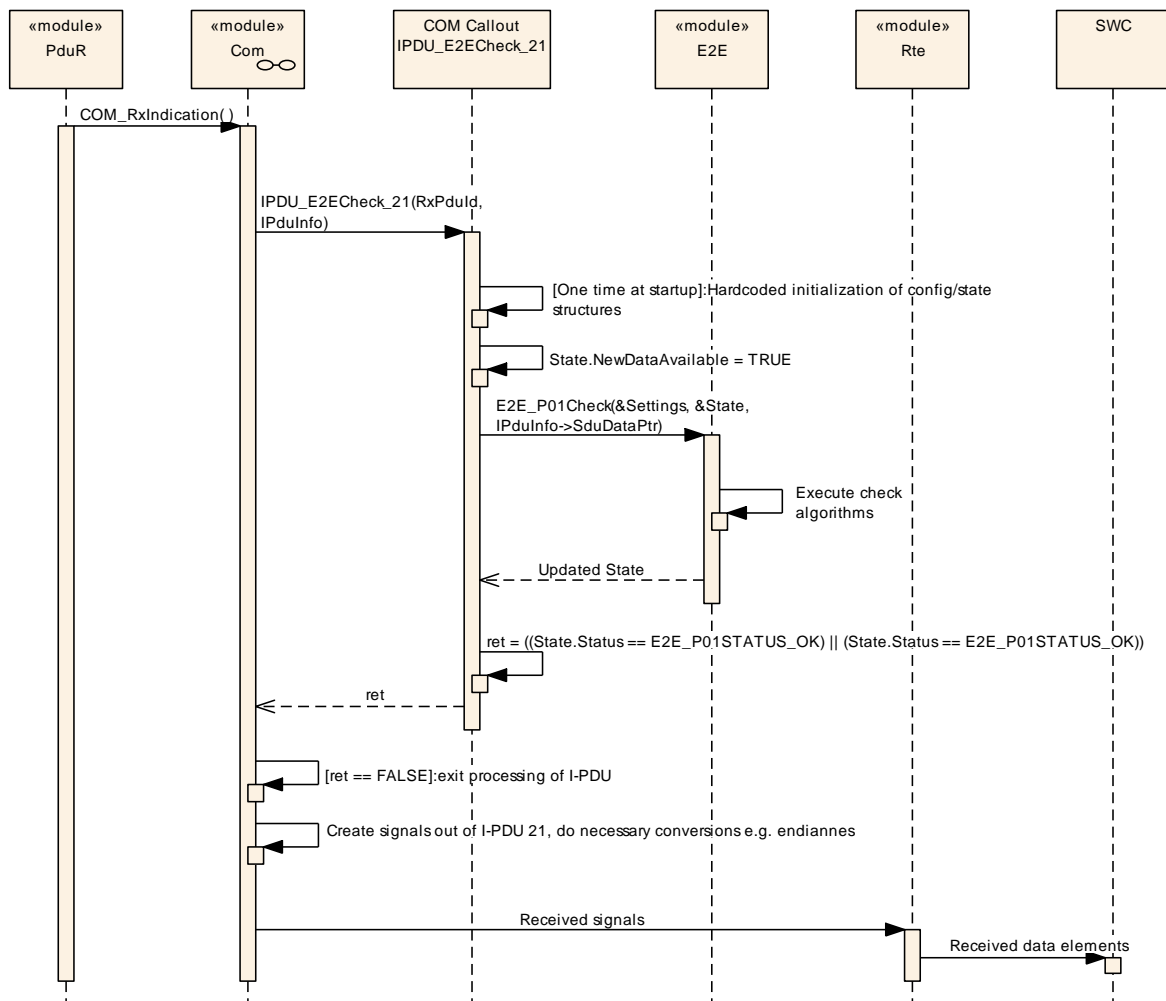


Figure 12-26: Callout - sequence - reception

[UC_E2E_00251] The reception callout for usage with E2E shall be the following:
`IPDU_E2ECheck_<IPDU ID>(PduIdType RxPduId, PduInfoType* PduInfoPtr).`

For example, the callout to protect the signal groups in an I-PDU with handle 21 shall have the name `IPDU_E2ECheck_21()`. (RS_E2E_08528)

12.2.2 Methodology

Note: Different releases of AUTOSAR have different names for COM classes. The text description below is generalized to fit to different releases, but the diagrams are slightly different (main differences are different names of classes and objects).

The information how each signal group needs to be protected (e.g. which E2E Profile, which offset) is defined in System Template [9], Software Component Template [8] and ECU configuration [10]. This configuration information is used to generate the callout functions.

By means of the settings defined by AUTOSAR templates, it is possible to generate the COM callouts for invoking the E2E Library.

The configuration is done in the following configuration areas:

1. Definition of I-PDUs (system template)
2. Definition of E2E settings (software component template)
3. Association of I-PDUs to E2E protection settings (system template).
4. Definition of I-PDU details (ECU configuration)

The four above steps are described in more details below.

First, according to System Template, the I-PDUs exchanged by COM are defined.

Secondly, according to Software Component Template, for each signal group to be protected, the classes `EndToEndProtection` and `EndToEndDescription` are defined. The settings include information like CRC offset.

Thirdly, according to System Template, each I-PDU to be protected is associated to a corresponding `EndToEndProtection`.

Fourth, after the extraction of ECU configuration, according to ECU configuration, the I-PDU handles (numerical I-PDU identifiers) and callout functions are defined. COM requires that there is a separate callout function for each I-PDU (separate piece of code).

All configuration options needed to generate the COM callouts automatically is available in AUTOSAR methodology. For each I-PDU to be protected/checked, a separate callout routine shall be generated, which invokes E2E Library (once or several times).

[UC_E2E_00270] The COM E2E callout shall be generated for the I-PDU for which the corresponding `EndToEnd*` metaclasses are defined. (RS_E2E_08528)

[UC_E2E_00290] If the E2EProtection is done via COM Callouts then the `EndToEndProtectionISignalPdu` shall be defined. (RS_E2E_08528)

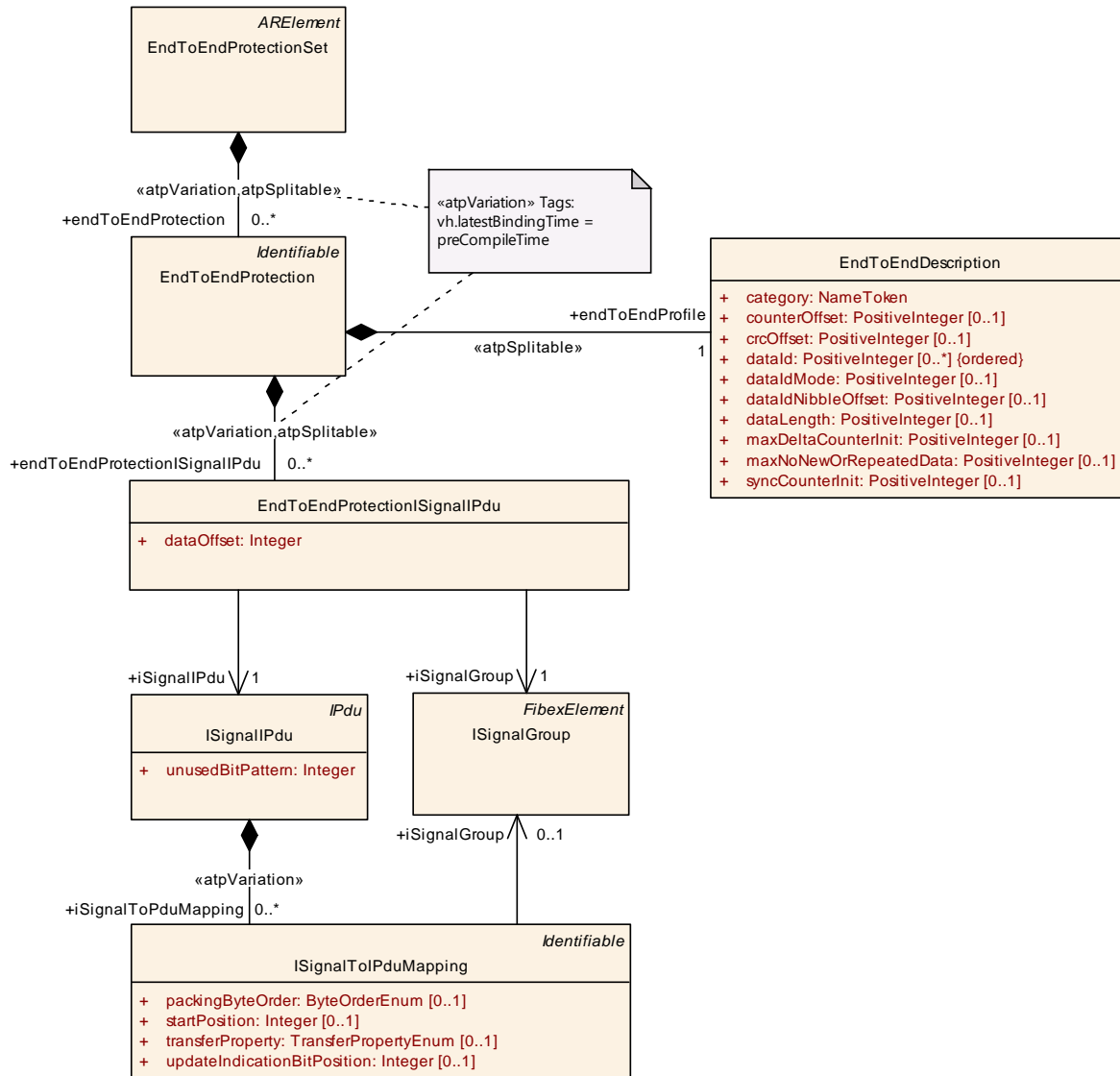


Figure 12-27: Release R4.0.1 and newer: COM Callouts Configuration (hardcopy from DOC_PduEndToEndProtection)

Note that in R3.2 (contrary to $\geq R4.0$), the ISignalIPdu is called “SignalIPdu” and it inherits the unusedBitPattern attribute from IPdu.

The important settings are:

1. ISignalIPdu (represents an I-PDU)
 - a. ISignalIPdu.unusedBitPattern: bits that are not used in an I-PDU,
2. ISignalToIPduMapping: describes the mapping of signals to I-PDUs,
 - a. ISignalToIPduMapping.startPosition: offset in bits of a signal in the I-PDU,
3. EndToEndProtectionISignalIPdu: association of one E2E protection to a one I-PDU and to one signal group,
 - a. EndToEndProtectionISignalIPdu.dataOffset: offset in bits of the signal group in the I-PDU.

ISignalPdu.unusedBitPattern is not used by E2E COM callouts, because they are set by COM and E2E COM callouts operate on the same buffers.

12.2.3 Code Example

Note that the code examples for the COM E2E callouts are for the case when there is one signal group in the I-PDU. In general, it is possible to have N signal groups in an I-PDU and M signal groups protected by E2E, where $0 \leq M \leq N$. In such a case, the callout invokes E2E Library functions M times (for each of the protected signal group).

Transmitter

```

FUNC(boolean, COM_APPL_CODE) IPDU_E2EProtect_21 (PduIdType RxPduId,
CONSTP2VAR (PduInfoType, AUTOMATIC, COM_VAR_NOINIT) PduInfoPtr){
    /* At first run, instantiate the structures and set the init
    Values*/
    static E2E_P01ConfigType Cfg_Write_21 =
        { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
    static E2E_P01ProtectStateType Sta_Write_21 = {0};

    Std_ReturnType ret = E2E_P01Protect(& Cfg_Write_21,
& Sta_Write_21,
                                     IPduInfo->SduDataPtr);
    /* return TRUE if no error in protect function */
    return (ret != 0);
}

```

Receiver

```

FUNC(boolean, COM_APPL_CODE) IPDU_E2ECheck_21 (PduIdType RxPduId,
CONSTP2CONST (PduInfoType, AUTOMATIC, COM_VAR_NOINIT) PduInfoPtr) {
    /* At first run, instantiate the structures and set the init
    values*/
    static E2E_P01ConfigType Cfg_Read_21 =
        { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
    static E2E_P01CheckStateType Sta_Read_21 =
        {0, 0, TRUE, FALSE, E2E_P01STATUS_NONEWDATA};
    /* If callout is invoked, this means that new data is available
    At COM */
    Sta_Read_21.NewDataAvailable = TRUE;
    Std_ReturnType ret = E2E_P01Check(Cfg_Read_21, Sta_Read_21,
IPduInfo->SduDataPtr);

    /* return TRUE if no error, possibly only some messages lost
    Within counter tolerance */
    if(ret == E2E_OK &&
        (Sta_Read_21.Status == E2E_P01STATUS_OK ||
         Sta_Read_21.Status == E2E_P02STATUS_OKSOMELOST) ) {

        return TRUE;
    }
    else {

```



```
        return FALSE;  
    }  
}
```

12.3 Provision of the Protection Wrapper Interface on a ECU with COM Callout solution

In case an ECU can provide a safe hardware, COM Layer and RTE, it is possible to integrate SWCs which require the E2E Protection Wrapper interfaces by using a direct mapping of E2E Wrapper interfaces to RTE interfaces and perform the E2E protection according to the "COM Callout" approach. By this approach compatibility between the two solutions "E2E Protection Wrapper" and "COM Callout" is achieved. This implies that the CRC and Ctr fields are not yet filled on RTE level in Tx direction. For Rx direction the CRC and Ctr on RTE level are already evaluated by COM and filled with status information and thus do not contain the PDU checksum and counter anymore.

Caveat: The E2E wrapper approach involves technologies that are not subjected to the AUTOSAR standard and is superseded by the superior E2E transformer approach (which is fully standardized by AUTOSAR). Hence, new projects (without legacy constraints due to carry-over parts) shall use the fully standardized E2E transformer approach.

12.4 Protection at RTE level through E2E Transformer

In this scenario, the RTE is considered safety-related. COM is QM. The RTE does the serialization of data elements into one dynamic-size signal, then RTE calls E2E to protect it. Then, RTE provides this E2E-protected dynamic-size signal to COM.

This solution is out-of-box, which means that AUTOSAR needs to be configured, but there is no need of integrator code for the E2E invocation.

This scenario is specified in details in SWS E2E Transformer.

13 Not applicable requirements

[SWS_E2E_NA_00294] These requirements are not applicable to this

specification. (SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007,
SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00158, SRS_BSW_00160,
SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00168,
SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00301, SRS_BSW_00302,
SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307,
SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00310, SRS_BSW_00312,
SRS_BSW_00314, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00325,
SRS_BSW_00327, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331,
SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00336,
SRS_BSW_00339, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343,
SRS_BSW_00346, SRS_BSW_00347, SRS_BSW_00348, SRS_BSW_00350,
SRS_BSW_00351, SRS_BSW_00353, SRS_BSW_00357, SRS_BSW_00358,
SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00361, SRS_BSW_00369,
SRS_BSW_00371, SRS_BSW_00373, SRS_BSW_00374, SRS_BSW_00375,
SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380,
SRS_BSW_00381, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00385,
SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390,
SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395,
SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399,
SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00402, SRS_BSW_00403,
SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00407,
SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411,
SRS_BSW_00412, SRS_BSW_00413, SRS_BSW_00414, SRS_BSW_00415,
SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422,
SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426,
SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432,
SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00439,
SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00447, SRS_BSW_00448,
SRS_BSW_00449, SRS_BSW_00450, SRS_BSW_00451, SRS_BSW_00452,
SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00456, SRS_BSW_00457,
SRS_BSW_00458, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461,
SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465,
SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470,
SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00473, SRS_BSW_00477,
SRS_BSW_00478, SRS_BSW_00479, SRS_BSW_00480, SRS_BSW_00481,
SRS_BSW_00482, SRS_BSW_00483, RS_E2E_08535, SRS_LIBS_00001,
SRS_LIBS_00002, SRS_LIBS_00003, SRS_LIBS_00004, SRS_LIBS_00005,
SRS_LIBS_00007, SRS_LIBS_00008, SRS_LIBS_00009, SRS_LIBS_00010,
SRS_LIBS_00011, SRS_LIBS_00012, SRS_LIBS_00013, SRS_LIBS_00015,
SRS_LIBS_00016, SRS_LIBS_00017, SRS_LIBS_00018, SRS_LIBS_08518,
SRS_LIBS_08521, SRS_LIBS_08525, SRS_LIBS_08526)