

Document Title	Specification of SW-C End-to-End Communication Protection Library
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	428
Document Classification	Standard
Document Version	3.2.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History			
Date	Version	Changed by	Change Description
28.02.2014	3.2.0	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
31.10.2013	3.1.0	AUTOSAR Release Management	<ul style="list-style-type: none"> • Correction in E2E variant 1C • Various minor corrections • Editorial changes
20.02.2013	3.0.0	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	Version	Changed by	Change Description
21.12.2011	2.0.0	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
19.10.2010	1.1.0	AUTOSAR Administration	<ul style="list-style-type: none"> • Corrected the wrapper configuration. • Corrected the code example for the usage of the wrapper.
17.12.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1	Introduction and functional overview	7
2	Acronyms and abbreviations	9
3	Related documentation.....	10
3.1	Input documents.....	10
3.2	Related standards and norms	11
4	Constraints and assumptions	12
4.1	Limitations	12
4.1.1	Limitations when invoking library at the level of data elements.....	12
4.2	Applicability to automotive domains	13
4.3	Background information concerning functional safety	13
4.3.1	Functional safety and communication.....	13
4.3.2	Sources of faults in E2E communication.....	14
4.3.3	Communication faults	15
4.4	Implementation of the E2E Library	16
5	Dependencies to/from other modules.....	17
5.1.1	Required file structure.....	17
5.1.2	Dependency on CRC library	18
6	Requirements traceability	19
7	Functional specification	22
7.1	Overview of communication protection.....	22
7.2	Overview of E2E Profiles.....	23
7.2.1	Error classification.....	24
7.2.2	Error detection	25
7.3	Specification of E2E Profile 1	26
7.3.1	Data Layout.....	27
7.3.2	Counter	27
7.3.3	Data ID.....	28
7.3.4	CRC calculation	29
7.3.5	Timeout detection	31
7.3.6	E2E Profile 1 variants	31
7.3.7	E2E_P01Protect	32
7.3.8	Calculate CRC	32
7.3.9	E2E_P01Check.....	33
7.4	Specification of E2E Profile 2	35
7.4.1	E2E_P02Protect	37
7.4.2	E2E_P02Check.....	40
7.5	Version Check.....	47
8	API specification.....	49
8.1	Imported types.....	49
8.2	Type definitions	49

8.2.1	E2E Profile 1 types	50
8.2.2	E2E Profile 2 types	55
8.3	Routine definitions.....	60
8.3.1	E2E Profile 1 routines	60
8.3.2	E2E Profile 2 routines	61
8.3.3	Elementary protocol routines	62
8.3.4	Auxiliary Functions.....	67
8.4	Call-back notifications	68
8.5	Scheduled functions.....	68
8.6	Expected Interfaces.....	68
8.6.1	Mandatory Interfaces	69
9	Sequence Diagrams for invoking E2E Library	70
9.1	Sender.....	70
9.1.1	Sender of data elements.....	70
9.1.2	Sender at sigal group level	72
9.2	Receiver	73
9.2.1	Receiver at data element level.....	75
9.2.2	Receiver at signal group level.....	77
10	Configuration specification.....	79
10.1	Published Information.....	79
11	Annex A: Safety Manual for usage of E2E Library.....	80
11.1	E2E profiles and their standard variants.....	80
11.2	E2E error handling	80
11.3	Maximal lengths of Data, communication buses	80
11.4	Methodology of usage of E2E Library	82
11.5	Configuration constraints on Data IDs.....	82
11.5.1	Data IDs.....	82
11.5.2	Double Data ID configuration of E2E Profile 1	83
11.5.3	Alternating Data ID configuration of E2E Profile 1	84
11.5.4	Nibble configuration of E2E Profile 1	84
11.6	Building custom E2E protocols.....	85
11.7	I-PDU Layout.....	85
11.7.1	Alignment of signals to byte limits.....	86
11.7.2	Unused bits.....	86
11.7.3	Byte order (Endianness)	87
11.7.4	Bit order.....	89
11.8	RTE configuration constraints for SW-C level protection.....	90
11.8.1	Communication model for SW-C level protection	90
11.8.2	Multiplicities for SW-C level protection.....	90
11.8.3	Explicit access	91
12	Annex B: Application hints on usage of E2E Library	92
12.1	E2E Protection Wrapper.....	93
12.1.1	Functional overview	94
12.1.2	Application scenario with Transmission Manager	95
12.1.3	Application scenario with E2E Manager and Conversion Manager ...	97
12.1.4	File structure.....	100

12.1.5	Methodology	102
12.1.6	Error classification	107
12.1.7	E2E Protection Wrapper routines	109
12.1.8	Code Example	124
12.2	COM E2E Callouts	142
12.2.1	Functional overview	142
12.2.2	Methodology	146
12.2.3	Code Example	149
12.3	Provision of the Protection Wrapper Interface on a ECU with COM Callout solution.....	150
12.4	Examples for the implementation of E2E protection concepts based on E2E-Library	150
12.4.1	Basic principles.....	151
12.4.2	Determination of the integrity of a communication channel within the receiver	152
13	Usage and generation of DataIDLists for E2E profile 2.....	154
13.1	Example A (persistent routing error).....	154
13.2	Example B (forbidden configuration)	156
13.3	Conclusion	157
13.4	DataIDList example	157
14	Not applicable requirements	168

1 Introduction and functional overview

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).

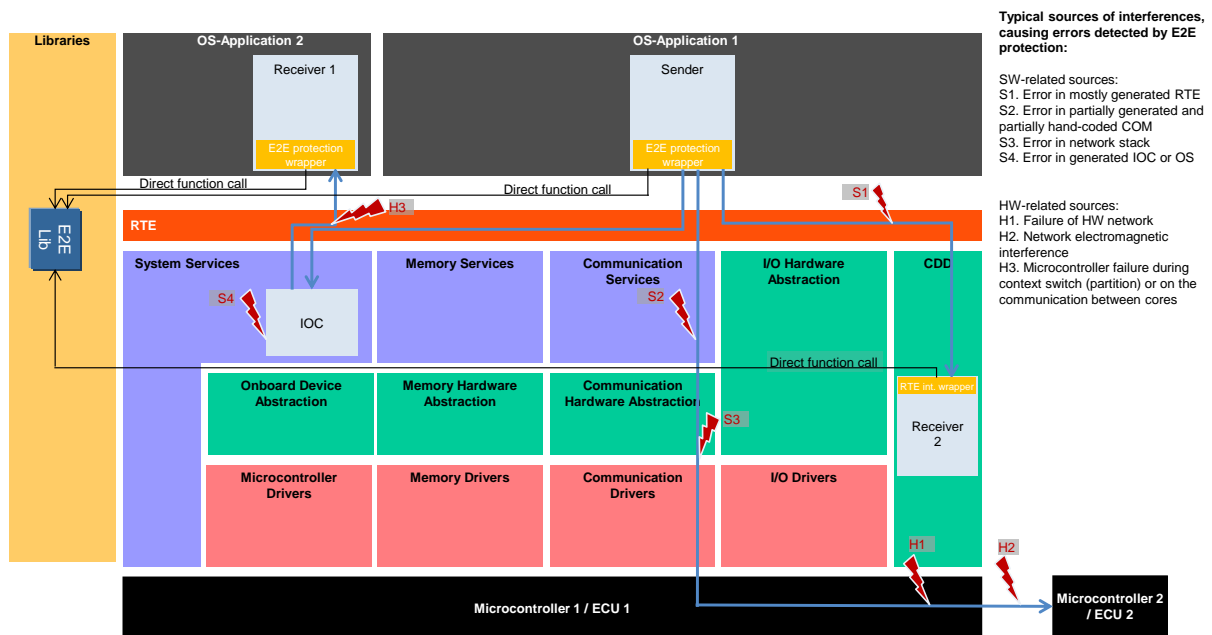


Figure 1-1: Example of faults mitigated by E2E protection

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).

Typically, the E2E protection is invoked from E2E Protection Wrapper or from 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 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).
2. 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 any subset of these data elements.

The 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 [10].

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.
Repetition	Repetition of information (see 4.3.3.1)
Loss	Loss of information (see 4.3.3.2)
Delay	Delay of information (see 4.3.3.3)
Insertion	Insertion of information (see 4.3.3.4)
Masquerade	Masquerade (see 4.3.3.5)
Incorrect addressing	Incorrect addressing of information (see 4.3.3.6).
Incorrect sequence	Incorrect sequence of information (see 4.3.3.7).
Corruption	Corruption of information (see 4.3.3.8).
Asymmetric information	Asymmetric information sent from a sender to multiple receivers (see 4.3.3.9)
Subset	Information from a sender received by only a subset of the receivers (see 4.3.3.10)
Blocking	Blocking access to a communication channel (see 4.3.3.11)

Table 2-1: Acronyms and abbreviations

In the whole document, there are many requirements that apply to all E2E Profiles at the same time (i.e. to E2E Profile 01 and E2E Profile 02). 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 two strings with 01 and 02 respectively instead of XX. For example, E2E_PXXCheck() develops to the following two E2E_P01Check(), E2E_P02Check().

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 COM
AUTOSAR_SWS_COM.pdf

- [5] Specification of BSW Scheduler
AUTOSAR_SWS_Scheduler.pdf

- [6] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

- [7] Specification of CRC Routines
AUTOSAR_SWS_CRCLibrary.pdf

- [8] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

- [9] Requirements on Libraries
AUTOSAR_SRS_Libraries.pdf

- [10] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf

- [11] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

- [12] System Template
AUTOSAR_TPS_SystemTemplate.pdf

- [13] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

3.2 Related standards and norms

[14] ISO 26262:2011
<http://www.iso.org/>

4 Constraints and assumptions

4.1 Limitations

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.

If a given sender-receiver communication is only intra-ECU (within microcontroller), then it is not defined within the configuration what the layout of the serialized Data shall be. On the other side, as the communication is intra-ECU, on both sides the software is probably generated by the same RTE generator, so the decision on the layout can be specific to the generator. It is recommended to serialize the data to have the CRC at the profile-specific position of the CRC and the Counter at the profile-specific position of the Counter (like for inter-ECU communication).

4.1.1 Limitations when invoking library at the level of data elements

[UC_E2E_00224][If the E2E Library is invoked at the level of data elements (e.g. from SW-Cs or from E2E Protection Wrapper), then the communication shall be an explicit sender-receiver communication, in 1:1 and 1:N multiplicities.]()

In other words, if E2E Library is invoked at the level of data elements, then N:1 multiplicity, implicit communication, and remaining communication models (in particular client-server model) are not supported.

[UC_E2E_00255][If the E2E Library is invoked at the level of data elements and 1:N communication model is used and the data elements are sent using more than one I-PDU, then all these I-PDUs shall have the same layout.]()

[UC_E2E_00226] [For each 1:N sender-receiver relationship the user of AUTOSAR shall define one specific layout to which the data elements that are going to be protected by E2E-Library are mapped for data transmission.]()

[UC_E2E_00326] [In case a user of AUTOSAR needs protected intra-ECU communication and protected inter-ECU communication to implement a safety-related sender-receiver relationship, the defined inter-ECU communication I-PDU layout shall be used for both transmissions.]()

If a user of AUTOSAR needs a protected intra-ECU communication to implement a safety-related sender-receiver relationship, then a specific layout (not restricted to the needs of COM I-PDUs) can be defined and used.

Currently AUTOSAR does not provide the functionality to describe and handle more than one layout for the same data element (e.g. within the RTE) by using different protection mechanisms depending on Intra-ECU and Inter-ECU communication. Thus, for a 1:N sender-receiver relationship the user of E2E-Library is responsible to select one appropriate layout for the to be protected data elements. E.g. for a 1:N sender-receiver relationship the COM I-PDU layout can be used for the transmission of data elements protected by E2E-Library to receivers located within and without the ECU.

4.2 Applicability to automotive domains

The library is applicable for the realization of safety-related automotive systems implemented by various SW-Cs distributed across different ECUs in a vehicle, interacting via communication links. The library may also be used for intra-ECU communication (e.g. between memory partitions or between CPU cores).

4.3 Background information concerning functional safety

This chapter provides some safety background information considered during the design of the E2E library, including the fault model for communication and definition of sources of faults.

4.3.1 Functional safety and communication

With respect to the exchange of information in safety-related systems, the mechanisms for the in-time detection of causes for faults or effects of faults as listed below can be used to design according safety concepts e.g. which achieve freedom from interference between system elements sharing a common communication infrastructure (see ISO 26262 [14] part 6, annex D.2.4):

- repetition of information;
- loss of information;

- delay of information;
- insertion of information;
- masquerade or incorrect addressing of information;
- incorrect sequence of information;
- corruption of information;
- asymmetric information sent from a sender to multiple receivers;
- information from a sender received by only a subset of the receivers;
- blocking access to a communication channel.

4.3.2 Sources of faults in E2E communication

E2E communication protection aims to detect and mitigate the causes for or effects of communication faults arising from:

1. (systematic) software faults,
2. (random) hardware faults,
3. transient faults due to external influences.

These three sources are described in the sections below.

4.3.2.1 Software faults

Software like communication stack modules and RTE may contain faults, which are of a systematic nature.

Systematic faults may occur in any stage of the system's life cycle including specification, design, manufacturing, operation, and maintenance, and they will always appear when the circumstances (e.g. trigger conditions for the root-cause) are the same. The consequences of software faults can be failures of the communication like interruption of sending of data, overrun of the receiver (e.g. buffer overflow), or underrun of the sender (e.g. buffer empty).

To prevent (or to handle) resulting failures the appropriate technical measures to detect and handle such faults (e.g. program flow monitoring or E2E) have to be considered.

4.3.2.2 Random hardware faults

A random hardware fault is typically the result of electrical overload, degradation, aging or exposure to external influences (e.g. environmental stress) of hardware parts. A random hardware fault cannot be avoided completely, but its probability can be evaluated and appropriate technical measures can be implemented (e.g. diagnostics).

4.3.2.3 External influences, environmental stress

This includes influences like EMI, ESD, humidity, corrosion, temperature or mechanical stress (e.g. vibration).

4.3.3 Communication faults

Relevant faults related to the exchange of information are listed in this section.

4.3.3.1 Repetition of information

A type of communication fault, where information is received more than once.

4.3.3.2 Loss of information

A type of communication fault, where information or parts of information are removed from a stream of transmitted information.

4.3.3.3 Delay of information

A type of communication fault, where information is received later than expected.

4.3.3.4 Insertion of information

A type of communication fault, where additional information is inserted into a stream of transmitted information.

4.3.3.5 Masquerading

A type of communication fault, where non-authentic information is accepted as authentic information by a receiver.

4.3.3.6 Incorrect addressing

A type of communication fault, where information is accepted from an incorrect sender or by an incorrect receiver.

4.3.3.7 Incorrect sequence of information

A type of communication fault, which modifies the sequence of the information in a stream of transmitted information.

4.3.3.8 Corruption of information

A type of communication fault, which changes information.

4.3.3.9 Asymmetric information sent from a sender to multiple receivers

A type of communication fault, where receivers do receive different information from the same sender.

4.3.3.10 Information from a sender received by only a subset of the receivers

A type of communication fault, where some receivers do not receive the information.

4.3.3.11 Blocking access to a communication channel

A type of communication fault, where the access to a communication channel is blocked.

4.4 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.] (SRS_LIBS_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 or E2E callouts) 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.] ()

5 Dependencies to/from other modules

5.1.1 Required file structure

The figure below shows the required structure of E2E library and required file inclusions.

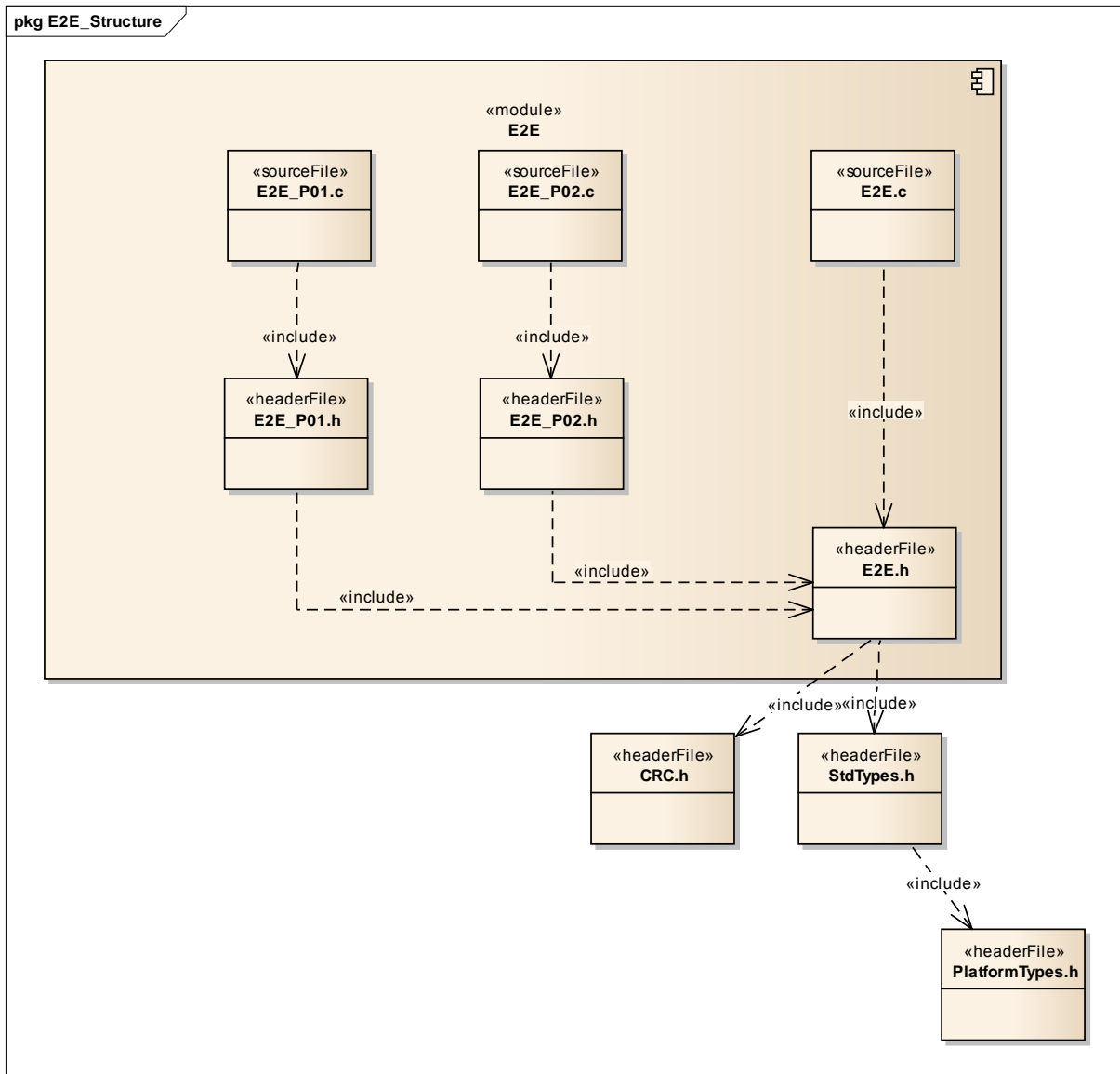


Figure 5-1: File dependencies

[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 and E2E_PXX.h (where XX: 01, 02).]()

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

[SWS_E2E_00111] [E2E.h shall exclusively include: CRC.h, Std_Types.h, PlatformTypes.h; E2E.h shall not include any other files.](SRS_BSW_00436)

[SWS_E2E_00112] [E2E.c shall exclusively include E2E.h.]()

[SWS_E2E_00113] [E2E_PXX.h shall exclusively include E2E.h.]()

[SWS_E2E_00114] [Each E2E_PXX.c file shall exclusively include the corresponding E2E_PXX.h file and it shall include the E2E_MemMap.h file.]()

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.]()

Note that as there are no configuration options in the E2E library, there is no E2E_Cfg.h file. Moreover, ComStack_Types.h are not needed by E2E, neither are RTE header files.

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_P02Check() 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 (see Figure 7-6) in >=R4.0 and R3.2.

6 Requirements traceability

Traceability table in R3.2.x:

Requirement	Satisfied by
-	E2E0123
-	E2E0128
-	E2EUSE087
-	E2E0129
-	E2E0143
-	E2EUSE0257
-	E2E0138
-	E2E0153
-	E2EUSE206
-	E2EUSE0250
-	E2E0130
-	E2E0288
-	E2EUSE230
-	E2E0083
-	E2EUSE235
-	E2E0196
-	E2EUSE062
-	E2EUSE0273
-	E2E0137
-	E2EUSE209
-	E2EUSE0277
-	E2EUSE089
-	E2E0226
-	E2EUSE072
-	E2E0114
-	E2E0293
-	E2E0076
-	E2EUSE0278
-	E2EUSE0301
-	E2E0195
-	E2E0113
-	E2E0169
-	E2EUSE173
-	E2EUSE0296
-	E2E0112
-	E2EUSE207
-	E2EUSE203
-	E2EUSE071
-	E2E0099
-	E2E0148
-	E2EUSE233
-	E2E0141
-	E2EUSE0297
-	E2EUSE0274
-	E2E0136
-	E2E0095
-	E2E0075
-	E2E0161
-	E2EUSE0271
-	E2EUSE0240
-	E2EUSE0265
-	E2E0134

-	E2E0092
-	E2E0107
-	E2E0048
-	E2EUSE208
-	E2E0139
-	E2EUSE073
-	E2EUSE0248
-	E2EUSE0292
-	E2E0160
-	E2EUSE056
-	E2E0276
-	E2EUSE204
-	E2EUSE237
-	E2E0140
-	E2E0287
-	E2E0200
-	E2EUSE0256
-	E2EUSE0300
-	E2EUSE202
-	E2E0125
-	E2E0190
-	E2EUSE055
-	E2EUSE0272
-	E2E0098
-	E2EUSE0249
-	E2E0132
-	E2EUSE0239
-	E2E0158
-	E2E0166
-	E2E0152
-	E2EUSE0264
-	E2E0215
-	E2E0154
-	E2E0255
-	E2E0115
-	E2E0119
-	E2E0126
-	E2E0094
-	E2EUSE0213
-	E2E0145
-	E2E0149
-	E2E0151
-	E2EUSE0275
-	E2E0228
-	E2EUSE0260
-	E2EUSE0290
-	E2EUSE236
-	E2E0012
-	E2EUSE0267
-	E2E0085
-	E2EUSE0241
-	E2E0135
-	E2EUSE057
-	E2E0021
-	E2EUSE0262
-	E2E0096
-	E2EUSE0289
-	E2EUSE0165
-	E2E0020
-	E2E0097

-	E2E0133
-	E2E0033
-	E2E0110
-	E2EUSE0270
-	E2EUSE0192
-	E2E0121
-	E2EUSE053
-	E2E0146
-	E2EUSE205
-	E2EUSE063
-	E2E0127
-	E2E0142
-	E2E0017
-	E2E0018
-	E2EUSE051
-	E2E0269
-	E2E0118
-	E2E0163
-	E2E0011
-	E2EUSE0263
-	E2E0150
-	E2E0224
-	E2E0106
-	E2EUSE0280
-	E2EUSE061
-	E2EUSE0268
-	E2E0147
-	E2EUSE0242
-	E2E0124
-	E2EUSE0261
-	E2EUSE0258
-	E2E0122
-	E2EUSE0251
-	E2EUSE0266
-	E2E0227
-	E2EUSE170
-	E2EUSE0279
-	E2EUSE0259
-	E2EUSE232
-	E2E0120
-	E2E0091
BSW003	E2E0032
BSW00323	E2E0047
BSW00336	E2E0294
BSW00337	E2E0047
BSW00338	E2E0049, E2E0294
BSW00339	E2E0294, E2E0216
BSW00369	E2E0049, E2E0294
BSW00375	E2E0294
BSW00435	E2E0294
BSW00436	E2E0111
BSW08527	E2E0050
BSW08528	E2E0217
BSW08529	E2E0219, E2E0218
BSW08531	E2E0221, E2E0117, E2E0070
BSW08533	E2E0219, E2E0218
BSW08534	E2E0047, E2E0022, E2E0214
BSW08536	E2E0082
BSW168	E2E0294

7 Functional specification

This chapter contains the specification of the internal functional behavior of the E2E Library. For general introduction of the E2E Library, see first Chapter 1.

7.1 Overview of communication protection

An important aspect of a communication protection mechanism is its standardization and its flexibility for different purposes. This is resolved by having a set of E2E Profiles, where each E2E Profile is configurable by function call parameters.

Each E2E Profile is non-generated, deterministic software code, where all inputs and settings are passed by function parameters. E2E Library functions are stateless and they are supposed to be invoked by SW-Cs (e.g. using a E2E protection wrapper, see Chapter 12.1.1), or from COM (e.g. by intermediary of COM E2E callouts, see Chapter 12.2).

Moreover, some E2E Profiles have standard E2E variants. An E2E variant is simply a set of configuration options to be used with a given E2E Profile. For example, in E2E Profile 1, the positions of CRC and counter are configurable. The E2E variant 1A requires that CRC starts at bit 0 and counter starts at bit 8.

Apart from E2E Profiles, the E2E Library provides also elementary functions (e.g. multibyte CRCs) to build additional (e.g. vendor-specific) safety protocols.

E2E protection uses the following safety mechanisms:

- Sender: addition of control fields like CRC or counter to the transmitted data;
- Receiver: evaluation of the control fields from the received data, calculation of control fields (e.g. CRC calculation on the received data), comparison of calculated control fields with an expected/received content.

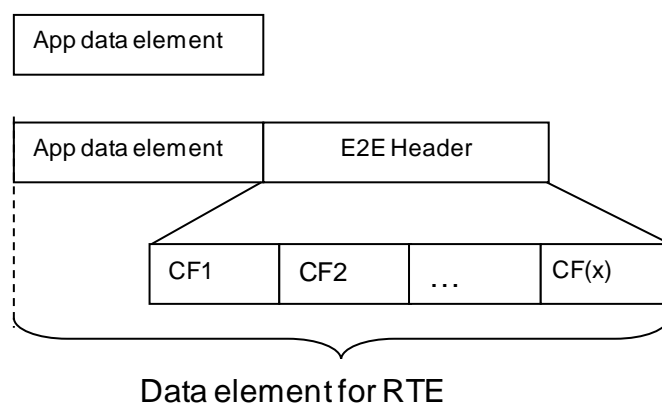


Figure 7-1: Safety protocol concept (with exemplary location of the E2E header)

Each E2E profile has a specific set of control fields with a specific functional behavior and with specific properties for the detection of communication faults.

7.2 Overview of E2E Profiles

The E2E profiles provide a consistent set of data protection mechanisms, designed to protecting against the faults considered in the fault model.

Each E2E profile provides an alternative way to protect the communication, by means of different algorithms. However, each E2E profile has almost identical API.

[SWS_E2E_00221] Each E2E Profile shall use a subset of the following data protection mechanisms:

1. A CRC, provided by CRC library;
2. A Sequence Counter incremented at every transmission request, the value is checked at receiver side for correct incrementation;
3. An Alive Counter incremented at every transmission request, the value checked at the receiver side if it changes at all, but correct incrementation is not checked;
4. A specific ID for every port data element sent over a port or a specific ID for every I-PDU group (global to system, where the system may contain potentially several ECUs);
5. Timeout detection:
 1. Receiver communication timeout;
 2. Sender acknowledgement timeout.

Depending on the used communication and network stack, appropriate subsets of these mechanisms are defined as E2E communication profiles.](SRS_LIBS_08531)

Some of above mechanisms are implemented in RTE, COM and/or communication stacks. However, to reduce or avoid an allocation of safety requirements to these modules, they are not considered: E2E Library provides all mechanisms internally (only with usage of CRC library).

The E2E Profiles can be used for both inter and intra ECU communication. The E2E Profiles are optimized for communication over CAN, FlexRay and can be used for LIN.

Depending on the system, the user selects which E2E Profile is to be used, from the E2E Profiles provided by E2E Library.

[SWS_E2E_00217] The implementation of the E2E Library shall provide at least one of the E2E Profiles.](SRS_LIBS_08528)

However, this is possible that specific implementations of E2E Library do not provide all two profiles, but only a one of them.

7.2.1 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_00338, 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.] ()

[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)

[SWS_E2E_00047] [The following error flags for errors shall be used by all E2E Library functions. The functions E2E_P01Protect(), E2E_P01Check(), E2E_P02Protect(), E2E_P02Check() shall use these values as return value:

Type or error or status	How do caller of E2E shall handle it	Related code	Value [hex]
At least one pointer parameter is a NULL pointer	Development error or Integration error	E2E_E_INPUTERR_NULL	0x13
At least one input parameter is erroneous, e.g. out of range	Development error or Integration error	E2E_E_INPUTERR_WRONG	0x17
An internal library error has occurred (e.g. error detected by program flow monitoring, violated invariant or postcondition)	Development error or Integration error	E2E_E_INTERR	0x19
Function completed successfully	N/A	E2E_E_OK	0x00

Table 7-1: Return values of E2E Library functions

](SRS_BSW_00337, SRS_BSW_00323, SRS_LIBS_08534)

There is no need that there is Hamming distance between error codes, as the codes are not transmitted over the bus.

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

SWS E2E does not provide any requirements on the extent of usage of program flow monitoring (e.g. quantity of checkpoints to use within). This is left to the implementer, which shall consider ISO 26262 requirements (e.g. table 4 from ISO 26262-6, which highly recommends control flow monitoring for ASIL C/D and recommends it for ASIL B). In case a specific implementation uses program flow monitoring, then the E2E_E_INTERR is to be used.

[UC_E2E_00313] [The caller of the E2E functions E2E_P01Protect() / E2E_P01Check() / E2E_P02Protect() / E2E_P02Check() shall handle the errors/states defined in SWS_E2E_00047 according to the column “How do caller of E2E shall handle it”.]()

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.

7.2.2 Error detection

[SWS_E2E_00012] [The internal library mechanisms shall detect and report errors shall be implemented according to the pre-defined E2E Profiles specified in sections 7.3 and 7.4.]()

7.3 Specification of E2E Profile 1

Profile 1 shall provide the following mechanisms:

[SWS_E2E_00218]

Mechanism	Description
Counter	4bit (explicitly sent) representing numbers from 0 to 14 incremented on every send request. Both Alive Counter and Sequence Counter mechanisms are provided by E2E Profile 1, evaluating the same 4 bits.
Timeout monitoring	Timeout is determined by E2E Library by means of evaluation of the Counter, by a non-blocking read at the receiver. Timeout is reported by E2E Library to the caller by means of the status flags in E2E_P01ReceiverStatusType.
Data ID	16 bit, unique number, included in the CRC calculation. For dataIdMode equal to 0, 1 or 2, the Data ID is not transmitted, but included in the CRC computation (implicit transmission). For dataIdMode equal to 3: <ul style="list-style-type: none"> the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits, the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data. the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for dataIDMode equal to 0, 1 or 2) .
CRC	CRC-8-SAE J1850 - $0x1D (x^8 + x^4 + x^3 + x^2 + 1)$, but with different start and XOR values (both start value and XOR value are 0x00). This CRC is provided by CRC library. Starting with AUTOSAR R4.0, the SAE8 CRC function of the CRC library uses 0xFF as start value and XOR value. To compensate a different behavior of the CRC library, the E2E Library applies additional XOR 0xFF operations starting with R4.0, to come up with 0x00 as start value and XOR value. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN.

Table 7-2: E2E mechanisms

The E2E mechanisms can detect the following faults or effects of faults:

E2E Mechanism	Detected communication faults
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Transmission on a regular	Loss, delay, blocking

basis and timeout monitoring using E2E-Library ¹⁾	
Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, Asymmetric information ²⁾
¹⁾ Implementation by sender and receiver, which are using E2E-Library	
²⁾ for a set of data protected by same CRC	

Table 7-3: Detectable communication faults using Profile 1

](SRS_LIBS_08529, SRS_LIBS_08533)

[SWS_E2E_00070] [E2E Profile 1 shall use the polynomial of CRC-8-SAE J1850, i.e. the polynomial $0x1D (x^8 + x^4 + x^3 + x^2 + 1)$, but with start value and XOR value equal to $0x00$.](SRS_LIBS_08531)

For details of CRC calculation, the usage of start values and XOR values see CRC Library [7]. Starting with R4.0, the SAE8 CRC function of the CRC library uses $0xFF$ as start value and XOR value. To compensate a different behavior of the CRC library, the E2E Library applies additional XOR $0xFF$ operations starting with R4.0, to come up with $0x00$ as start value and XOR value. Moreover, starting with R4.0, the SAE8 CRC function has an additional parameter `Crc_IsFirstCall`, which introduces a slightly different algorithm in E2E Profile 1 functions.

7.3.1 Data Layout

In the E2E Profile 1, the layout is in general free to be defined by the user – it is only constrained by the byte alignment user requirements [E2E0062](#) and [E2E0063](#) (i.e. bytes of data elements / signals must be aligned to byte limits). However, the E2E Profile 1 variants constrain the layout, see Chapter 7.3.6.

7.3.2 Counter

In E2E Profile 1, the counter is initialized, incremented, reset and checked by E2E profile.

[SWS_E2E_00075] [In E2E Profile 1, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request (from sender SW-C). When the counter reaches the value 14 ($0xE$), then it shall restart with 0 for the next send request (i.e. value $0xF$ shall be skipped). All these actions shall be executed by E2E Library.]()

[SWS_E2E_00076] [In E2E Profile 1, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following shall be detected: (1) no new data has arrived since last invocation of E2E library check function, (2) no new data has arrived since receiver start, (3) the data is repeated (4) counter is incremented by one (i.e. no data lost), (5) counter is incremented more

than by one, but still within allowed limits (i.e. some data lost), (6) counter is incremented more than allowed (i.e. too many data lost). All these actions shall be executed by E2E Library.]()

Case 3 corresponds to the failed alive counter check, and case 6 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

7.3.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

[SWS_E2E_00163] [There shall be following four inclusion modes for the two-byte Data ID into the calculation of the one-byte CRC:

1. E2E_P01_DATAID_BOTH: both two bytes (double ID configuration) are included in the CRC, first low byte and then high byte (see variant 1A - [SWS E2E 00227](#)) or
2. E2E_P01_DATAID_ALT: depending on parity of the counter (alternating ID configuration) the high and the low byte is included (see variant 1B - [SWS E2E 00228](#)). For even counter values the low byte is included and for odd counter values the high byte is included.
3. E2E_P01_DATAID_LOW: only the low byte is included and high byte is never used. This equals to the situation if the Data IDs (in a given application) are only 8 bits.
4. E2E_P01_DATAID_NIBBLE:
 - the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,
 - the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.
 - the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for the inclusion modes _BOTH, _ALT and _LOW)]()

[SWS_E2E_00085] [In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_BOTH or E2E_P01_DATAID_ALT the length of the Data ID shall be 16 bits (i.e. 2 byte).]()

[SWS_E2E_00169] [In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_LOW, the high byte of Data ID shall be set to 0x00.]()

The above requirement means that when high byte of Data ID is unused, it is set to 0x00.

[SWS_E2E_00306] [In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_NIBBLE, the high nibble of the high byte shall be 0x0.]()

The above requirement means that the address space with E2E_P01_DATAID_NIBBLE is limited to 12 bits.

In case of usage of E2E Library for protecting data elements, due to multiplicity of communication (1:1 or 1:N), a receiver of a data element receives it only from one sender. In case of usage of E2E Library for protecting I-PDUs, because each I-PDU has a unique Data ID, the receiver COM of an I-PDU receives it from only from one sender COM. As a result (regardless if the protection is at data element level or at I-PDUs), the receiver expects data with only one Data ID. The receiver uses the expected Data ID to calculate the CRC. If CRC matches, it means that the Data ID used by the sender and expected Data ID used by the receiver are the same.

7.3.4 CRC calculation

E2E Profile 1 uses CRC-8-SAE J1850, but using different start and XOR values. This checksum is already provided by AUTOSAR CRC library, which typically is quite efficient and may use hardware support.

[SWS_E2E_00083][E2E Profile 1 shall use CRC-8-SAE J1850 for CRC calculation. It shall use 0x00 as the start value and XOR value.]()

[SWS_E2E_00190][E2E Profile 1 shall use the Crc_CalculateCRC8 () function of the SWS CRC Library for calculating CRC checksums.]()

Note: The CRC used by E2E Profile 1 is different than the CRCs used by FlexRay and CAN and is provided by different software modules (FlexRay and CAN CRCs are provided by hardware support in Communication Controllers, not by CRC library).

The CRC calculation is illustrated by the following two examples.

For standard variant 1A:

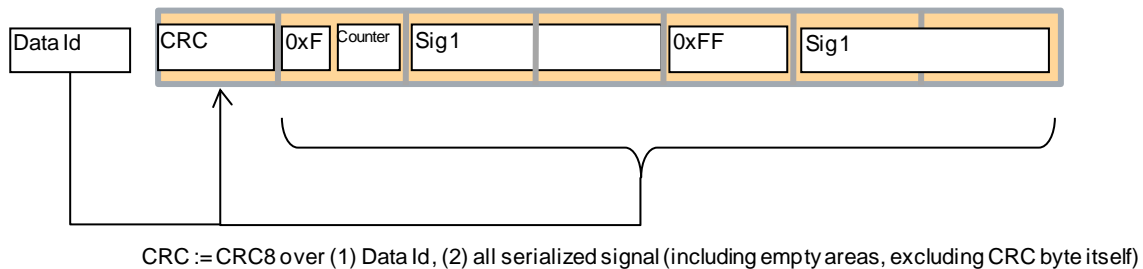


Figure 7-2: E2E Profile 1 variant 1A CRC calculation example

For standard variant 1C:

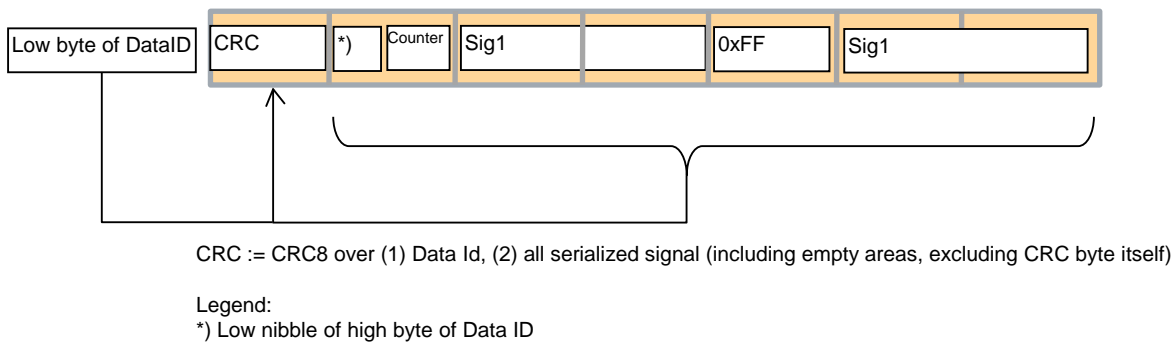


Figure 7-3: E2E Profile 1 variant 1C CRC calculation example

The Data ID can be encoded in CRC in different ways, see [SWS E2E 00163](#).

[SWS_E2E_00082][In E2E Profile 1, the CRC is calculated over:
1. First over the one or two bytes of the Data ID (depending on Data ID configuration) and then
2. Over all transmitted bytes of a safety-related complex data element/signal group (except the CRC byte).](SRS_LIBS_08536)

7.3.5 Timeout detection

The previously mentioned mechanisms (CRC, counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively signal groups, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts. The independent execution of the receiver is required by [E2EUSE0089](#).

The attribute State->Status = E2E_P01STATUS_REPEATED means that there is a repetition (caused either by communication loss, delay or duplication of the previous message). The receiver uses State->Status for detecting communication timeouts.

7.3.6 E2E Profile 1 variants

The E2E Profile 1 has variants. The variants are specific configurations of E2E Profile.

[SWS_E2E_00227] [The E2E Profile variant 1A is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_BOTH
4. SignalPdu.unusedBitPattern = 0xFF.]()

[SWS_E2E_00228] [The E2E Profile variant 1B is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_ALTERNATING
4. SignalPdu.unusedBitPattern = 0xFF.]()

Below is an example compliant to 1A/1B:

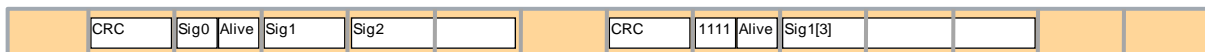


Figure 7-4: E2E Profile 1 example layout (two signal groups protected by E2E in one I-PDU)

[SWS_E2E_00307] [The E2E Profile variant 1C is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1st byte (i.e. starts with bit offset 12)
4. E2E_P01DataIDMode = E2E_P01_DATAID_NIBBLE
5. SignalPdu.unusedBitPattern = 0xFF.]()

7.3.7 E2E_P01Protect

[SWS_E2E_00195][The function E2E_P01Protect() shall:

1. write the Counter in Data,
2. write DataID nibble in Data (E2E_P01_DATAID_NIBBLE) in Data
3. compute the CRC over DataID and Data
4. write CRC in Data
5. increment the Counter (which will be used in the next invocation of E2E_P01Protect()),

as specified by Figure 7-5 and Figure 7-6.]()

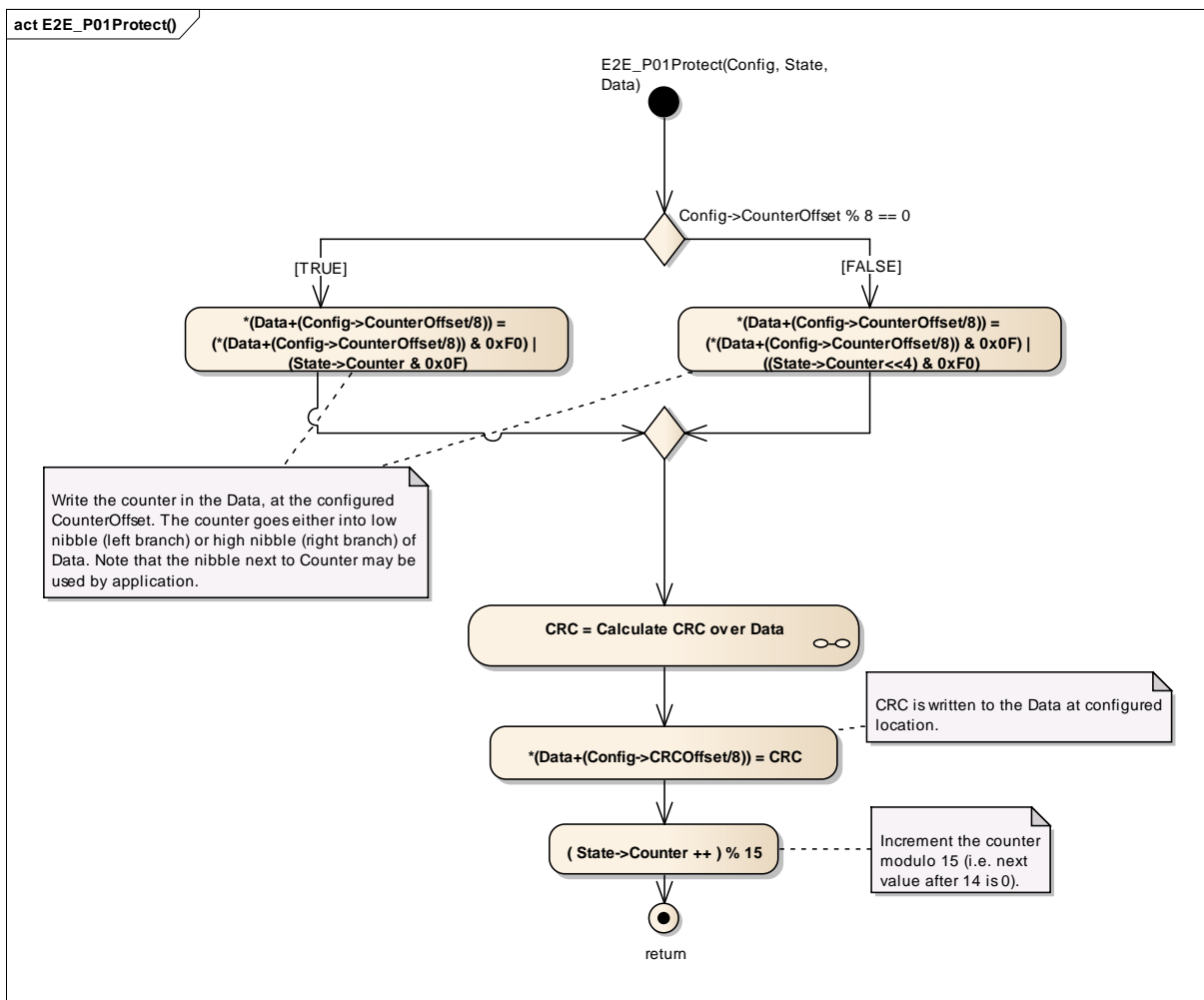


Figure 7-5: E2E_P01Protect()

7.3.8 Calculate CRC

The diagram of the function `E2E_P01Protect()` (see above chapter) and `E2E_P01Check()` (see below chapter) have a sub-diagram specifying the calculation of CRC:

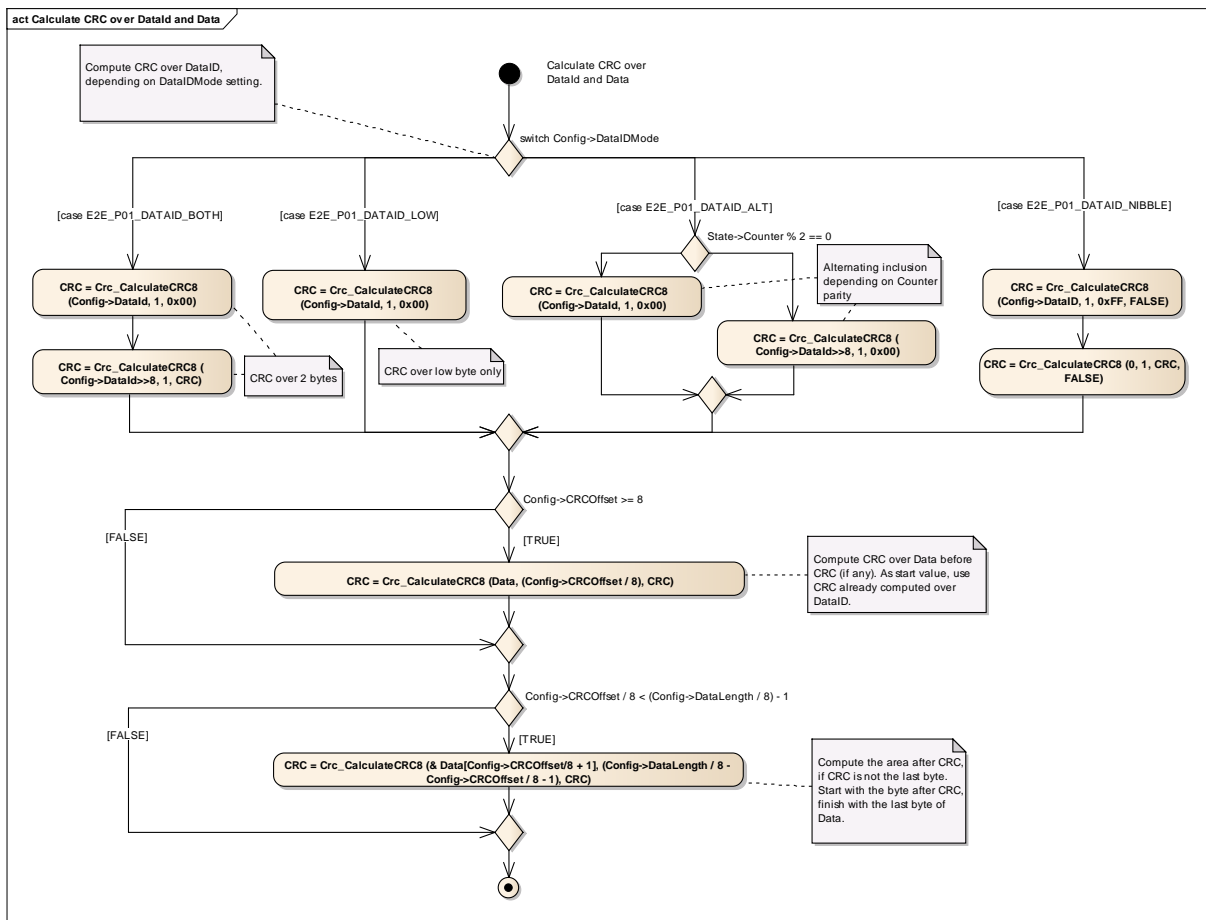


Figure 7-6: Subdiagram “Calculate CRC over Data ID and Data”, used by E2E_P01Protect() and E2E_P01Check()

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:

3. There is an additional parameter Crc_IsFirstCall
4. 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_P02Check() 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 (see Figure 7-6) in >=R4.0 and R3.2. This means Figure 7-6 is different in >=R4.0 and R3.2.

7.3.9 E2E_P01Check

[SWS_E2E_00196] The function E2E_P01Check shall

1. Check the CRC

2. Check the Data ID nibble, i.e. compare the expected value with the received value (for E2E_P01_DATAID_NIBBLE configuration only)
3. Check the Counter,
4. determine the check Status,

as specified by Figure 7-7 and Figure 7-6.]()

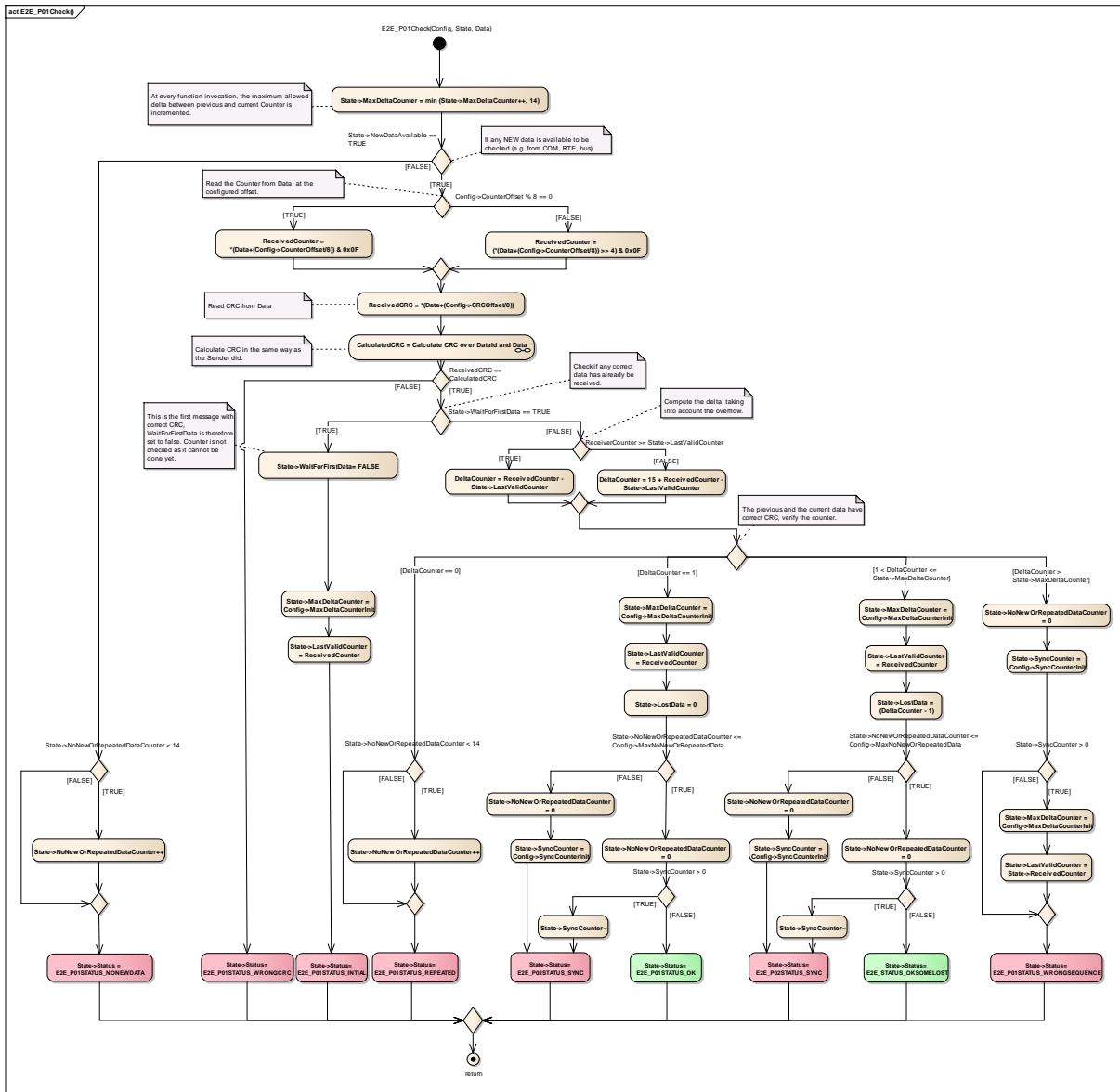


Figure 7-7: E2E_P01Check()

The diagram of the function E2E_P01Check() has a sub-diagram specifying the calculation of CRC, which is shown by Figure 7-6.

7.4 Specification of E2E Profile 2

[SWS_E2E_00219] [Profile 2 shall provide the following mechanisms:

Mechanism	Description
Sequence Number (Counter)	4bit (explicitly sent) representing numbers from 0 to 15 incremented by 1 on every send request (Bit 0:3 of Data[1]) at sender side. The counter is incremented on every call of the E2E_P02Protect() function, i.e. on every transmission request of the SW-C
Message Key used for CRC calculation (Data ID)	8 bit (not explicitly sent) The specific Data ID used to calculate the CRC depends on the value of the Counter and is an element of a pre-defined set of Data IDs (value of the counter as index to select the particular Data ID used for the protection). For every Data element, the List of Data IDs depending on each value of the counter is unique.
Safety Code (CRC)	8 bit explicitly sent (Data[0]) Polynomial: $0x2F (x^8 + x^5 + x^3 + x^2 + x + 1)$ Start value: $0xFF$ Final XOR-value: $0xFF$ Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay and CAN.

Table 7-4: Mechanisms of E2E Profile 2

](SRS_LIBS_08529, SRS_LIBS_08533)

The mechanisms provided by Profile 2 enable the detection of the relevant failure modes except message delay (for details see table 6):

Since this profile is implemented in a library, the library's E2E_P02Check() function itself cannot ensure to be called in a periodic manner. Thus, a required protection mechanism against undetected message delay (e.g. Timeout) must be implemented in the caller.

The E2E mechanisms can detect the following faults or effects of faults:

E2E Mechanism	Detected communication faults
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Transmission on a regular bases and timeout monitoring using E2E-Library ¹⁾	Loss, delay, blocking
Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, Asymmetric information ²⁾
¹⁾ Implementation by sender and receiver	
²⁾ for a set of data protected by same CRC	

Table 7-5: Detectable communication faults using Profile 2

[SWS_E2E_00117] [E2E Profile 2 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Library for calculating CRC checksums.] (SRS_LIBS_08531)

[SWS_E2E_00118] [E2E Profile 2 shall use 0xFF as the start value CRC_StartValue8 for CRC calculation.] ()

[SWS_E2E_00119] [In E2E Profile 2, the specific Data ID used to calculate a specific CRC shall be of length 8 bit.] ()

[SWS_E2E_00120] [In E2E Profile 2, the specific Data ID used for CRC calculation shall be selected from a pre-defined DataIDList[16] using the value of the Counter as an index.] ()

Each data, which is protected by a CRC owns a dedicated DataIDList which is deposited on the sender site and all the receiver sites.

The pre-defined DataIDList[16] is generated offline. In general, there are several factors influencing the contents of DataIDList, e.g.:

1. length of the protected data
2. number of protected data elements
3. number of cycles within a masquerading fault has to be detected
4. number of senders and receivers
5. characteristics of the CRC polynomial.

An example DataIDList is presented in Chapter 13.4.

Due to the limited length of the 8bit polynomial, a masquerading fault cannot be detected in a specific cycle when evaluating a received CRC value. Due to the adequate Data IDs in the DataIDList, a masquerading fault can be detected in one of the successive communication cycles.

Due to the underlying rules for the DataIDList, the system design of the application has to take into account that a masquerading fault is detected not until evaluating a certain number of communication cycles.

[SWS_E2E_00121] [In E2E Profile 2, the layout of the data buffer (Data) shall be as depicted in Figure 7-8 with a maximum length of 256 bytes (i.e. N=255)]

Data[0]	Data[1]	Data[2]	Data[N-1]	Data[N]
8 CRC	8 8 Counter	8 ... 8	8 ... 8	8 ... 8

Figure 7-8: Data of E2E Profile 2

] ()

[SWS_E2E_00122] [In E2E Profile 2, the CRC shall be Data[0].] ()

[SWS_E2E_00123][In E2E Profile 2, the Counter shall be the low nibble (Bit 0...Bit 3) of Data[1].]()

[SWS_E2E_00124][In E2E Profile 2, the E2E_P02Protect() function shall not modify any bit of Data except the bits representing the CRC and the Counter.]()

[SWS_E2E_00125][In E2E Profile 2, the E2E_P02Check() function shall not modify any bit in Data.]()

7.4.1 E2E_P02Protect

The E2E_P02Protect() function of E2E Profile 2 is called by a SW-C in order to protect its application data against the failure modes as shown in Table 7-5. E2E_P02Protect() therefore calculates the Counter and the CRC and puts it into the data buffer (Data). A flow chart with the visual description of the function E2E_P02Protect() is depicted in Figure 7-9 and Figure 7-10.

[SWS_E2E_00126][In E2E Profile 2, the E2E_P02Protect() function shall perform the activities as specified in Figure 7-9 and Figure 7-10.]()

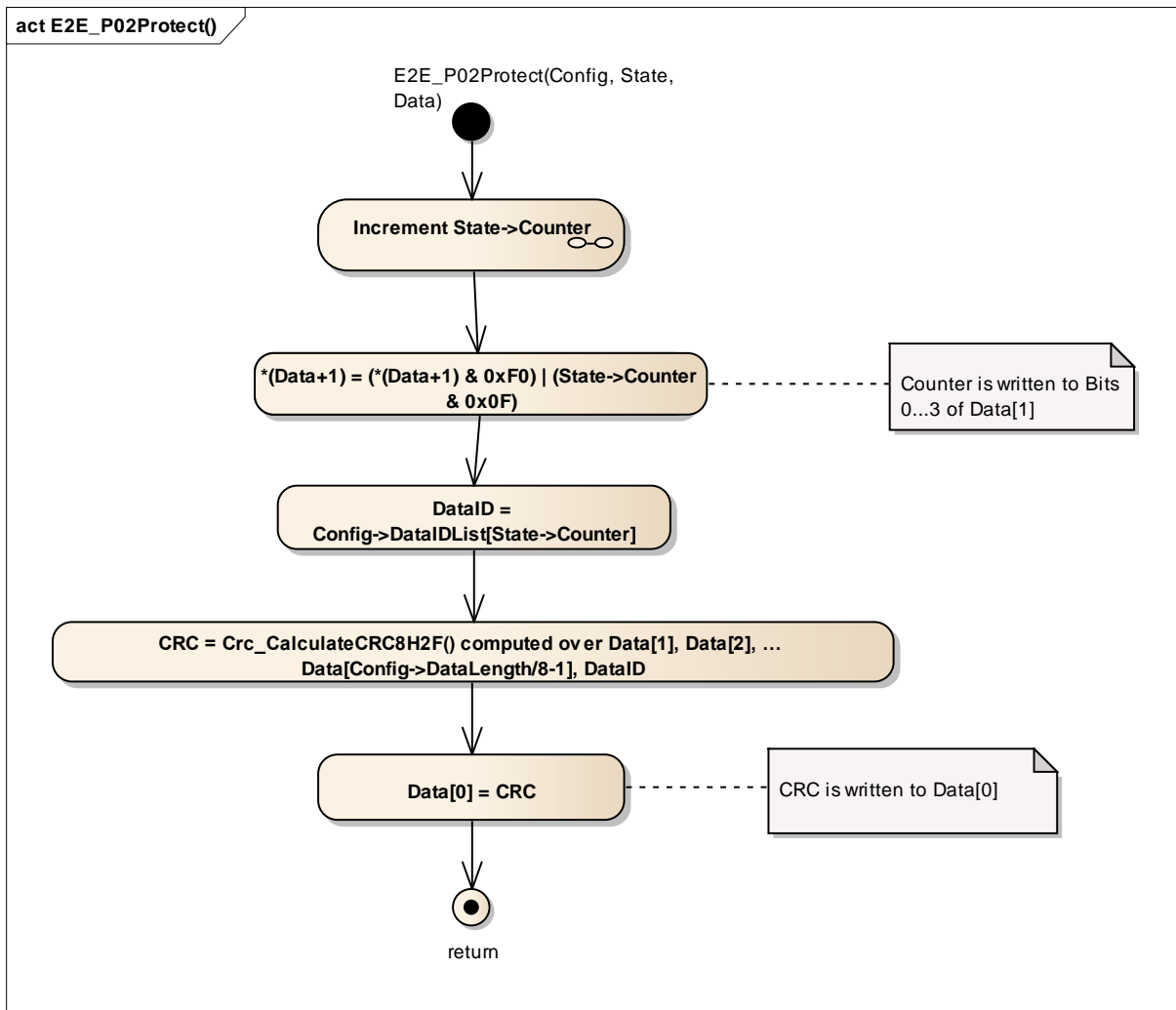


Figure 7-9: E2E_P02Protect()

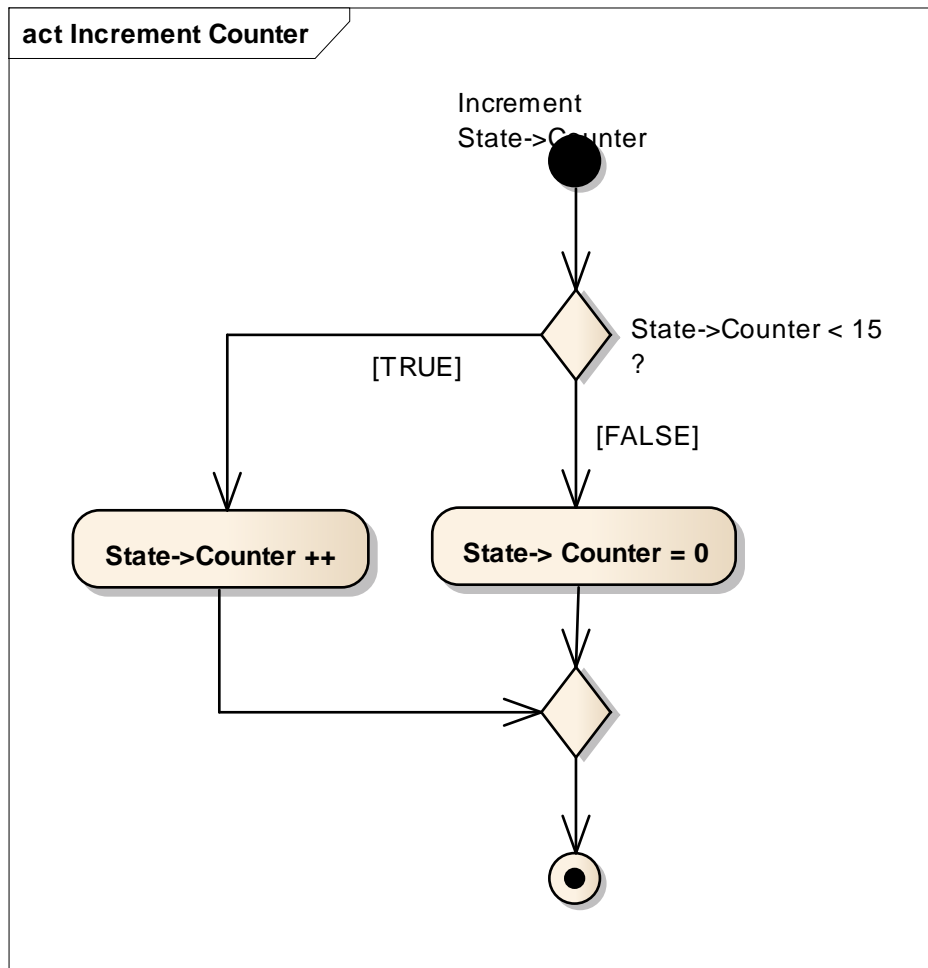


Figure 7-10: Increment Counter

[SWS_E2E_00127] [In E2E Profile 2, the E2E_P02Protect() function shall increment the Counter of the state (P02SenderStateType) by 1 on every transmission request from the sending SW-C, i.e. on every call of E2E_P02Protect().]()

[SWS_E2E_00128] [In E2E Profile 2, the range of the value of the Counter shall be [0...15].]()

[SWS_E2E_00129] [When the Counter has reached its upper bound of 15 (0xF), it shall restart at 0 for the next call of the E2E_P02Protect() from the sending SW-C.]()

[SWS_E2E_00130] [In E2E Profile 2, the E2E_P02Protect() function shall update the Counter (i.e. low nibble (Bit 0...Bit 3) of Data byte 1) in the data buffer (Data) after incrementing the Counter.]()

The specific Data ID used for this send request is then determined from a DataIDList[] depending on the value of the Counter (Counter is used as an index to select the Data ID from DataIDList[]). The DataIDList[] is defined in E2E_P02ConfigType.

[SWS_E2E_00132][In E2E Profile 2, after determining the specific Data ID, the E2E_P02Protect() function shall calculate the CRC over Data[1], Data[2], ... Data[Config->DataLength/8-1] of the data buffer (Data) extended with the Data ID.]()

[SWS_E2E_00133][In E2E Profile 2, the E2E_P02Protect() function shall update the CRC (i.e. Data[0]) in the data buffer (Data) after computing the CRC.]()

The specific Data ID itself is not transmitted on the bus. It is just a virtual message key used for the CRC calculation.

7.4.2 E2E_P02Check

The E2E_P02Check() function is used as an error detection mechanism by a caller in order to check if the received data is correct with respect to the failure modes mentioned in the profile summary.

A flow chart with the visual description of the function E2E_P02Check() is depicted in Figure 7-11 Figure 7-12 and Figure 7-13.

[SWS_E2E_00134][In E2E Profile 2, the E2E_P02Check() function shall perform the activities as specified in Figure 7-11, Figure 7-12and Figure 7-13.]()

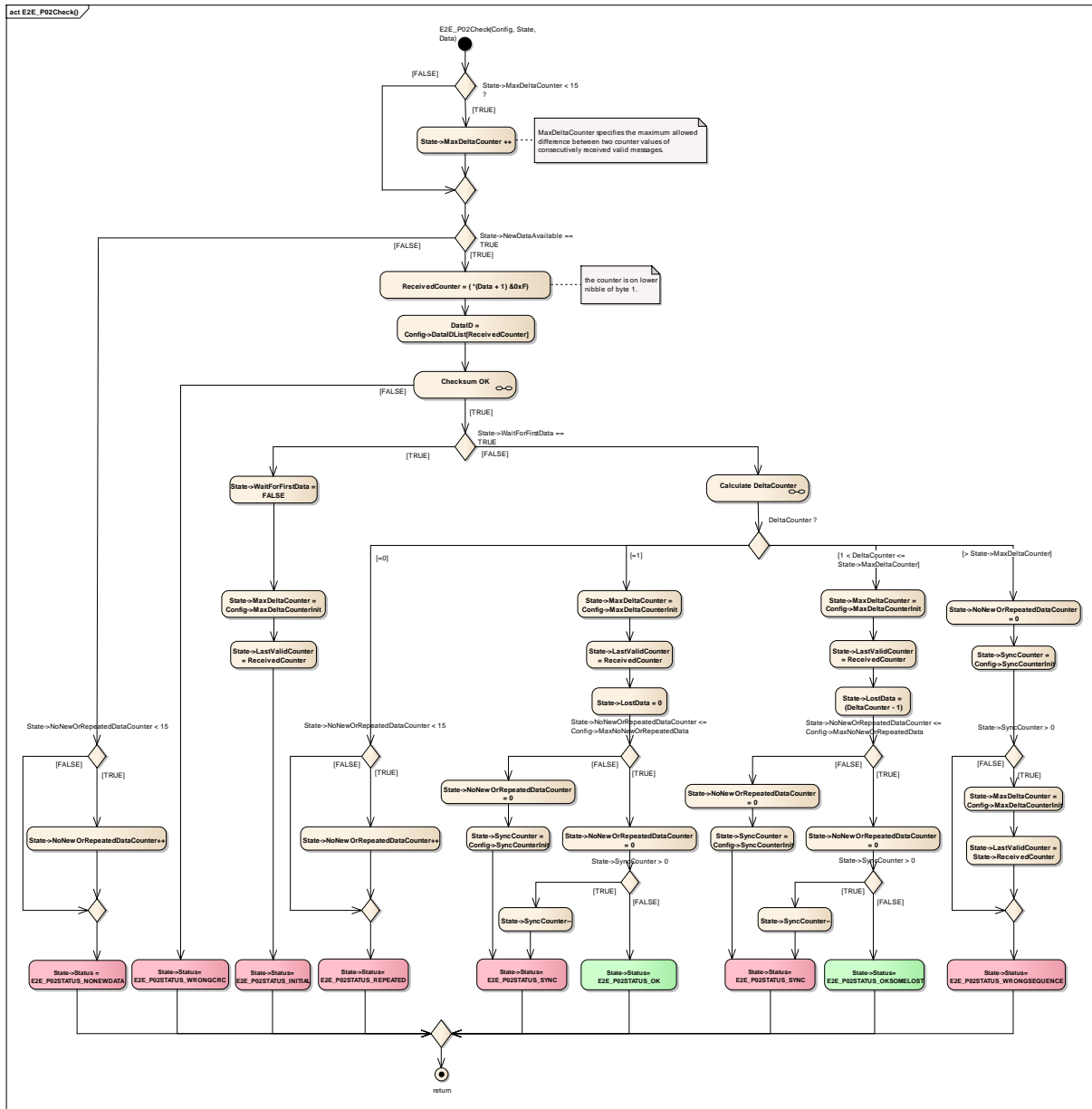


Figure 7-11: E2E_P02Check()

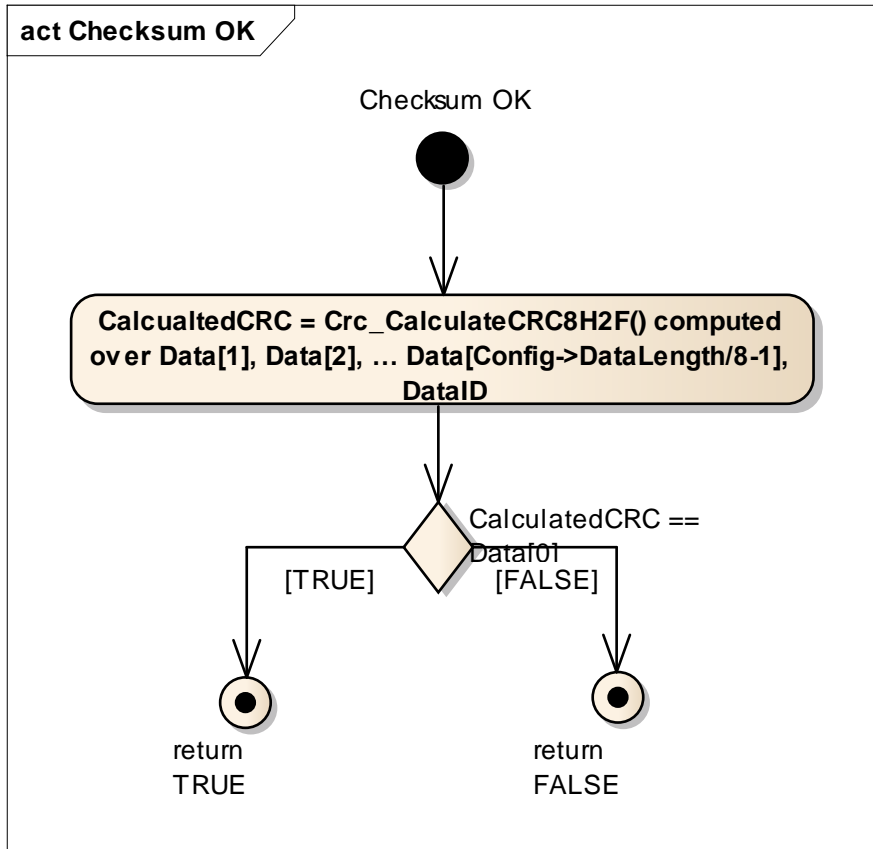


Figure 7-12: Checksum OK

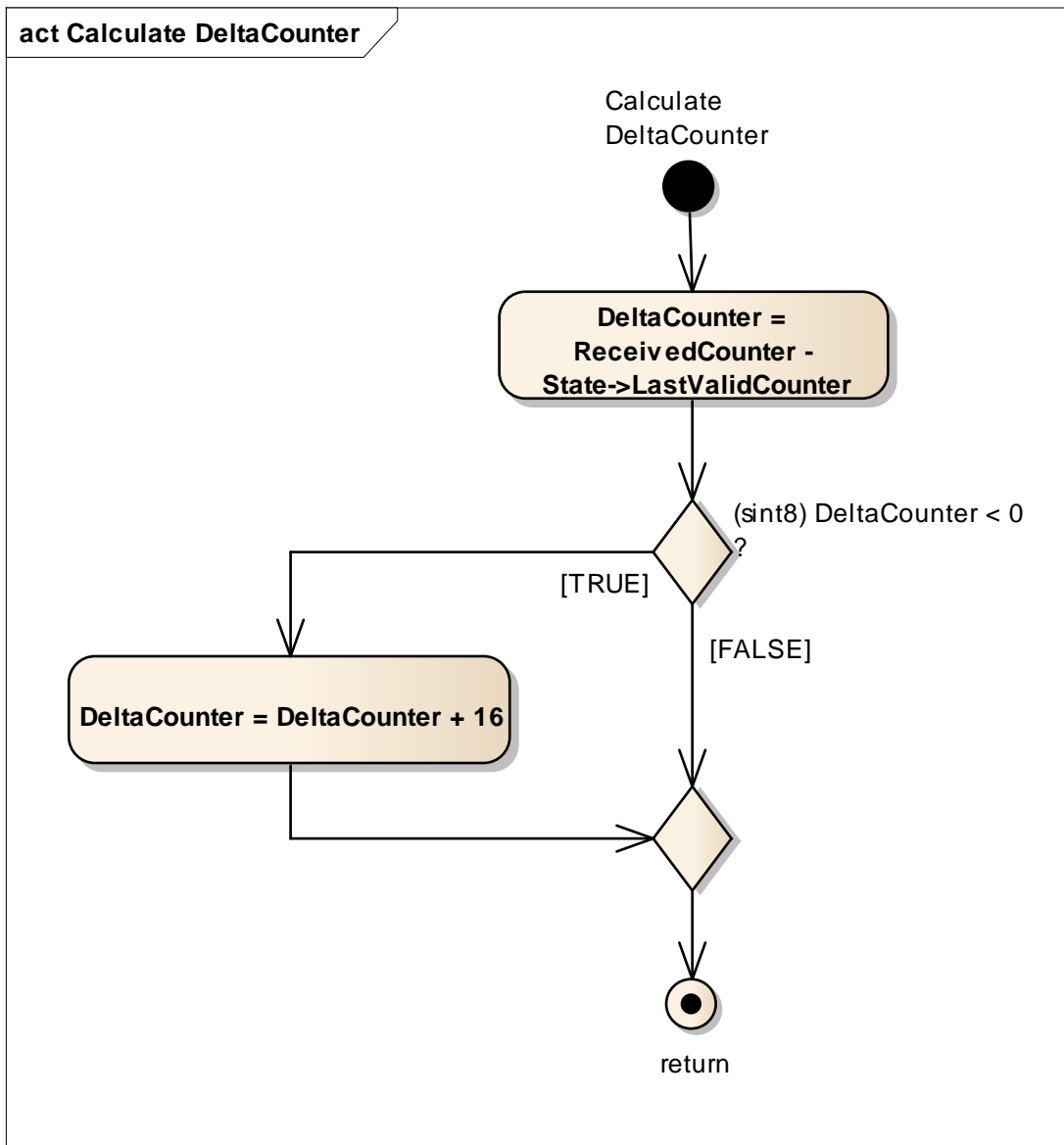


Figure 7-13: Calculate DeltaCounter

First, the E2E_P02Check() function increments the value MaxDeltaCounter. MaxDeltaCounter specifies the maximum allowed difference between two Counter values of two consecutively received valid messages.

Note: MaxDeltaCounter is used in order to perform a plausibility check for the failure mode re-sequencing.

If the flag NewDataAvailable is set, the E2E_P02Check() function continues with the evaluation of the CRC. Otherwise, it returns with Status set to E2E_P02STATUS_NONEWDATA.

To evaluate the correctness of the CRC, the following actions are performed:

- The specific Data ID is determined using the value of the Counter as provided in Data.
- Then the CRC is calculated over Data payload extended with the Data ID as last Byte:

CalculatedCRC = Crc_CalculateCRC8H2F() calculated over Data[1], Data[2], ... Data[Config->DataLength/8-1], Data ID

- Finally, the check for correctness of the received Data is performed by comparing CalculatedCRC with the value of CRC stored in Data.

In case CRC in Data and CalculatedCRC do not match, the E2E_P02Check() function returns with Status E2E_P02STATUS_WRONGCRC, otherwise it continues with further evaluation steps.

The flag WaitForFirstData specifies if the SW-C expects the first message after startup or after a timeout error. This flag should be set by the SW-C if the SW-C expects the first message e.g. after startup or after reinitialization due to error handling. This flag is allowed to be reset by the E2E_P02Check() function only. The reception of the first message is a special event because no plausibility checks against previously received messages is performed.

If the flag WaitForFirstData is set by the SW-C, E2E_P02Check() does not evaluate the Counter of Data and returns with Status E2E_P02STATUS_INITIAL. However, if the flag WaitForFirstData is reset (the SW-C does not expect the first message) the E2E_P02Check() function evaluates the value of the Counter in Data.

For messages with a received Counter value within a valid range, the E2E_P02Check() function returns either with E2E_P02STATUS_OK or E2E_P02STATUS_OKSOMELOST. In LostData, the number of missing messages since the most recently received valid message is provided to the SW-C.

For messages with a received Counter value outside of a valid range, E2E_P02Check() returns with one of the following states: E2E_P02STATUS_WRONGSEQUENCE or E2E_P02STATUS_REPEATED.

[SWS_E2E_00135][In E2E Profile 2, the local variable DeltaCounter shall be calculated by subtracting LastValidCounter from Counter in Data, considering an overflow due to the range of values [0...15].]()

Details on the calculation of DeltaCounter are depicted in Figure 7-13.

[SWS_E2E_00136] [In E2E Profile 2, MaxDeltaCounter shall specify the maximum allowed difference between two Counter values of two consecutively received valid messages.]()

[SWS_E2E_00137] [In E2E Profile 2, MaxDeltaCounter shall be incremented by 1 every time the E2E_P02Check() function is called, up to the maximum value of 15 (0xF).]()

[SWS_E2E_00138] [In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_NONEWDATA if the attribute NewDataAvailable is FALSE.]()

[SWS_E2E_00139] [In E2E Profile 2, the E2E_P02Check() function shall determine the specific Data ID from DataIDList using the Counter of the received Data as index.]()

[SWS_E2E_00140] [In E2E Profile 2, the E2E_P02Check() function shall calculate CalculatedCRC over Data[1], Data[2], ... Data[Config->DataLength/8-1] of the data buffer (Data) extended with the determined Data ID.]()

[SWS_E2E_00141] [In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_WRONGCRC if the calculated CalculatedCRC value differs from the value of the CRC in Data.]()

[SWS_E2E_00142] [In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_INITIAL if the flag WaitForFirstData is TRUE.]()

[SWS_E2E_00143] [In E2E Profile 2, the E2E_P02Check() function shall clear the flag WaitForFirstData if it returns with Status E2E_P02STATUS_INITIAL.]()

For the first message after start up no plausibility check of the Counter is possible. Thus, at least a minimum number of messages need to be received in order to perform a check of the Counter values and in order to guarantee that at least one correct message was received.

[SWS_E2E_00145] [The E2E_P02Check() function shall

- set Status to E2E_P02STATUS_WRONGSEQUENCE; and
- re-initialize SyncCounter with SyncCounterInit

if the calculated value of DeltaCounter exceeds the value of MaxDeltaCounter.]()

[SWS_E2E_00146] [The E2E_P02Check() function shall set Status to E2E_P02STATUS_REPEATED if the calculated DeltaCounter equals 0.]()

[SWS_E2E_00147] The E2E_P02Check() function shall set Status to E2E_P02STATUS_OK if the following conditions are true:

- the calculated DeltaCounter equals 1; and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. $\text{State} \rightarrow \text{NoNewOrRepeatedDataCounter} \leq \text{Config} \rightarrow \text{MaxNoNewOrRepeatedData}$); and
- the SyncCounter equals 0.]()

[SWS_E2E_00298] The E2E_P02Check() function shall

- re-initialize SyncCounter with SyncCounterInit; and
- set Status to E2E_P02STATUS_SYNC;

if the following conditions are true:

- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. $1 \leq \text{DeltaCounter} \leq \text{MaxDeltaCounter}$); and
- the value of the NoNewOrRepeatedDataCounter exceeds MaxNoNewOrRepeatedData. (i.e. $\text{State} \rightarrow \text{NoNewOrRepeatedDataCounter} > \text{Config} \rightarrow \text{MaxNoNewOrRepeatedData}$)])()

[SWS_E2E_00299] The E2E_P02Check() function shall

- decrement SyncCounter by 1; and
- set Status to E2E_P02STATUS_SYNC

if the following conditions are true:

- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. $1 \leq \text{DeltaCounter} \leq \text{MaxDeltaCounter}$); and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. $\text{State} \rightarrow \text{NoNewOrRepeatedDataCounter} \leq \text{Config} \rightarrow \text{MaxNoNewOrRepeatedData}$); and
- the SyncCounter exceeds 0.]()

[SWS_E2E_00148] The E2E_P02Check() function shall set Status to E2E_P02STATUS_OKSOMELOST if the following conditions are true:

- the calculated DeltaCounter is greater-than 1 but less-than or equal to MaxDeltaCounter (i.e. $1 < \text{DeltaCounter} \leq \text{MaxDeltaCounter}$); and
- the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. $\text{State} \rightarrow \text{NoNewOrRepeatedDataCounter} \leq \text{Config} \rightarrow \text{MaxNoNewOrRepeatedData}$); and
- the SyncCounter equals 0.]()

[SWS_E2E_00149] The E2E_P02Check() function shall set the value LostData to (DeltaCounter – 1) if the calculated DeltaCounter is greater-than 1 but less-than or equal to MaxDeltaCounter.]()

[SWS_E2E_00150] The E2E_P02Check() function shall re-initialize MaxDeltaCounter with MaxDeltaCounterInit if it returns one of the following Status:

- E2E_P02STATUS_OK; or

- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_INITIAL; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE on condition that SyncCounter exceeds 0 (i.e. SyncCounter > 0).]()

[SWS_E2E_00151] [The E2E_P02Check() function shall set LastValidCounter to Counter of Data if it returns one of the following Status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_INITIAL; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE on condition that SyncCounter exceeds 0 (i.e. SyncCounter > 0).]()

[SWS_E2E_00300] [The E2E_P02Check() function shall reset the NoNewOrRepeatedDataCounter to 0 if it returns one of the following status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE]()

[SWS_E2E_00301] [The E2E_P02Check() function shall increment NoNewOrRepeatedDataCounter by 1 if it returns the Status E2E_P02STATUS_NONEWDATA or E2E_P02STATUS_REPEATED up to the maximum value of Counter (i.e. 15 or 0xF).]()

7.5 Version Check

[

[SWS_E2E_00287] [This requirement applies for AUTOSAR release 3.2.x: Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_AR_MINOR_VERSION),
arPatchVersion (<Module>_AR_PATCH_VERSION),

swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_SW_MINOR_VERSION),
swPatchVersion (<Module>_SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template.]()

8 API specification

This chapter specifies the API of E2E Library.

8.1 Imported types

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

[SWS_E2E_00017]Imported types

<i>Module</i>	<i>Imported Type</i>
GENERIC TYPES	<InType>
RTE	Rte_Instance
Std_Types	Std_ReturnType
	Std_VersionInfoType

]()

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).

Because CRC and counter fit to 1 byte, there is no issue of byte order (endianness). Moreover, different CPU-specific bit order is also irrelevant.

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

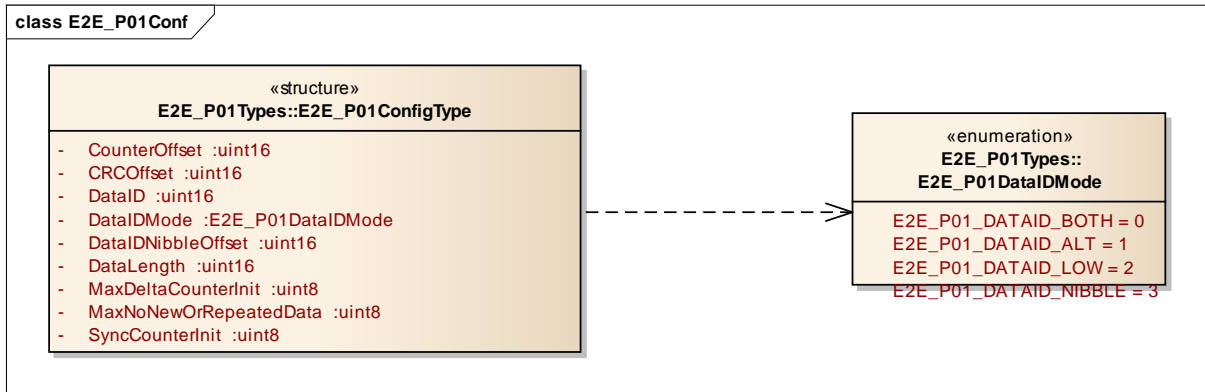


Figure 8-1: E2E Profile 1 configuration

8.2.1.1 E2E_P01ConfigType

[SWS_E2E_00018]E2E_P01ConfigType

Name:	E2E_P01ConfigType		
Type:	Structure		
Element:	uint16	CounterOffset	Bit offset of Counter in MSB first order. In variants 1A and 1B, CounterOffset is 8. The offset shall be a multiple of 4.
	uint16	CRCOffset	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.
	uint16	DataID	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'.
	uint16	DataIDNibbleOffset	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).

	E2E_P01DataIDMode	DataIDMode	Inclusion mode of ID in CRC computation (both bytes, alternating, or low byte only of ID included).
	uint16	DataLength	Length of data, in bits. The value shall be a multiple of 8 and shall be ≤ 240.
	uint8	MaxDeltaCounterInit	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.
	uint8	MaxNoNewOrRepeatedData	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	uint8	SyncCounterInit	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.		

]()

8.2.1.2 E2E_P01DataIDMode

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

[SWS_E2E_00200]E2E_P01DataIDMode

Name:	E2E_P01DataIDMode	
Type:	Enumeration	
Range:	E2E_P01_DATAID_BOTH	Two bytes are included in the CRC (double ID configuration) This is used in variant 1A.
	E2E_P01_DATAID_ALT	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 variant 1B.
	E2E_P01_DATAID_LOW	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	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 three inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.	

]()

8.2.1.3 E2E_P01SenderStateType

[SWS_E2E_00020]E2E_P01SenderStateType

Name:	E2E_P01SenderStateType		
Type:	Structure		
Element:	uint8	Counter	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 an invalid value. The four high bits are always 0.
Description:	State of the sender for a Data protected with E2E Profile 1.		

]()

8.2.1.4 E2E_P01ReceiverStateType

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.

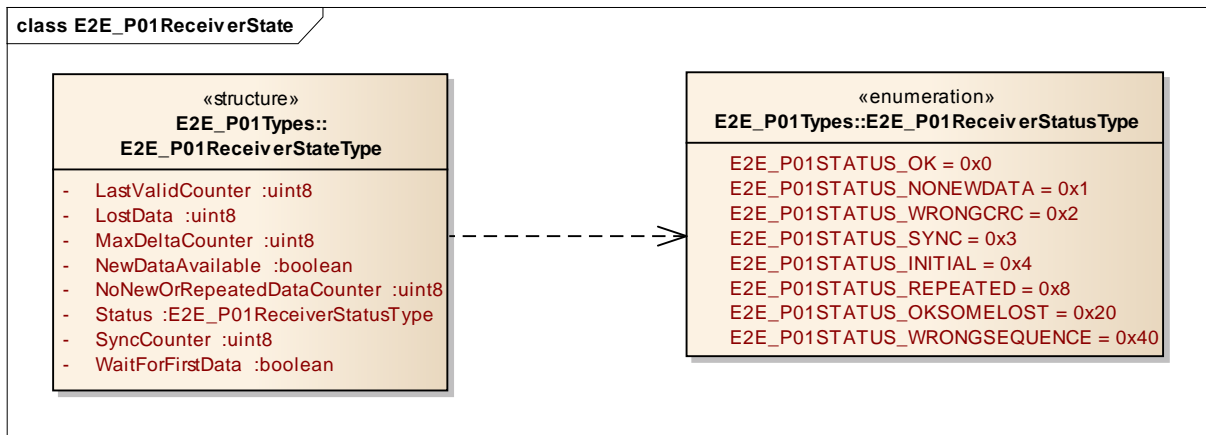


Figure 8-2: E2E Profile 1 receiver state

[SWS_E2E_00021]E2E_P01ReceiverStateType

Name:	E2E_P01ReceiverStateType		
Type:	Structure		
Element:	uint8	LastValidCounter	Counter value most recent received. If no data has been y received, then the value is 0x After each reception, the counter updated with the value received.
	uint8	MaxDeltaCounter	MaxDeltaCounter specifies th maximum allowed differenc between two counter values consecutively received val messages.
	boolean	WaitForFirstData	If true means that no correct da (with correct Data ID and CRC) ha been yet received after the receiv initialization or reinitialization.
	boolean	NewDataAvailable	Indicates to E2E Library that a ne data is available for Library to b checked. This attribute is set by th E2E Library caller, and not by th E2E Library.
	uint8	LostData	Number of data (messages) lo since reception of last valid on This attribute is set only if Statu equals E2E_P01STATUS_OK or E2E_P01STATUS_OKSOMELOS For other values of Status, th value of LostData is undefined.
	E2E_P01ReceiverStatusType	Status	Result of the verification of th Data, determined by the Chec function.
	uint8	SyncCounter	Number of Data required f validating the consistency of th counter that must be received wi a valid counter (i.e. counter with the allowed lock-in range) after th detection of an unexpecte behavior of a received counter.

	uint8	NoNewOrRepeatedDataCounter	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.			

⌋()

8.2.1.5 E2E_P01ReceiverStatusType

[SWS_E2E_00022]E2E_P01ReceiverStatusType

Name:	E2E_P01ReceiverStatusType	
Type:	Enumeration	
Range:	E2E_P01STATUS_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.
	E2E_P01STATUS_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.
	E2E_P01STATUS_WRONGCRC	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_P01DataIDMode = E2E_P01_DATAID_NIBBLE). The two above errors can be a result of corruption, incorrect addressing or masquerade.
	E2E_P01STATUS_SYNC	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	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	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	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.
	E2E_P01STATUS_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 <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.
Description:	Result of the verification of the Data in E2E Profile 1, determined by the Check function.	

](SRS_LIBS_08534)

8.2.2 E2E Profile 2 types

8.2.2.1 E2E_P02ConfigType

[SWS_E2E_00152]E2E_P02ConfigType

Name:	E2E_P02ConfigType		
Type:	Structure		
Element:	uint16	DataLength	Length of Data, in bits. The value shall be a multiple of 8.
	uint8[16]	DataIDList	An array of appropriately chosen Data IDs for protection against masquerading.
	uint8	MaxDeltaCounterInit	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.
	uint8	MaxNoNewOrRepeatedData	The maximum amount of missing or repeated Data which the receiver

			does not expect to exceed under normal communication conditions.
	uint8	SyncCounterInit	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:	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.		

]()

8.2.2.2 E2E_P02SenderStateType

[SWS_E2E_00153]E2E_P02SenderStateType

Name:	E2E_P02SenderStateType		
Type:	Structure		
Element:	uint8	Counter	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 2.		

]()

8.2.2.3 E2E_P02ReceiverStateType

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

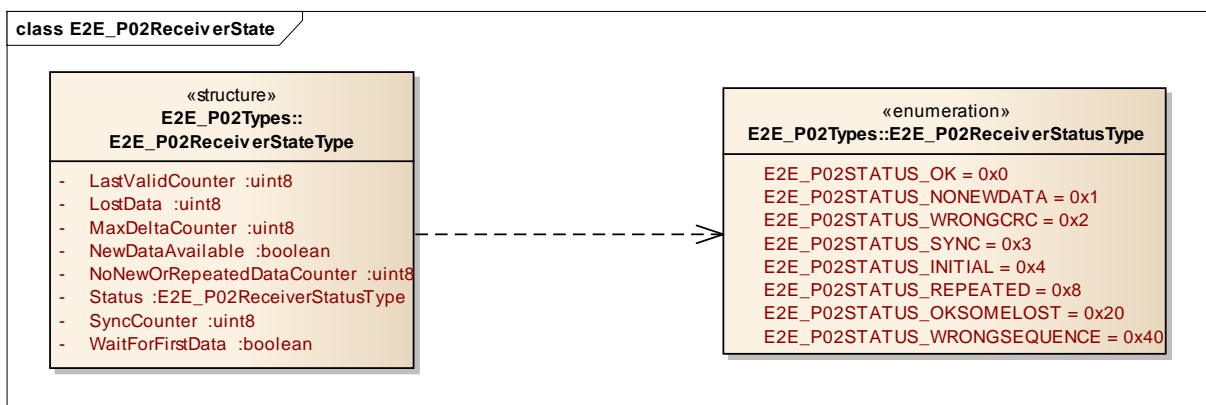


Figure 8-3: E2E Profile 2 receiver state

[SWS_E2E_00154]E2E_P02ReceiverStateType

Name:	E2E_P02ReceiverStateType
Type:	Structure

Element:	uint8	LastValidCounter	Counter of last valid received message.
	uint8	MaxDeltaCounter	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
	boolean	WaitForFirstData	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	boolean	NewDataAvailable	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.
	uint8	LostData	Number of data (messages) lost since reception of last valid one.
	E2E_P02ReceiverStatusType	Status	Result of the verification of the Data, determined by the Check function.
	uint8	SyncCounter	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.
	uint8	NoNewOrRepeatedDataCounter	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.		

]()

8.2.2.4 E2E_P02ReceiverStatusType

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

[SWS_E2E_00214]E2E_P02ReceiverStatusType

Name:	E2E_P02ReceiverStatusType	
Type:	Enumeration	
Range:	E2E_P02STATUS_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.
	E2E_P02STATUS_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.
	E2E_P02STATUS_WRONGCRC	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P02STATUS_SYNC	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	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	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	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by

		<p>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>
	E2E_P02STATUS_WRONGSEQUENCE	<p>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>
Description:	Result of the verification of the Data in E2E Profile 2, determined by the Check function.	

](SRS_LIBS_08534)

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

The specified routines in several cases may call each other. For example, a profile routine from 8.3.1 may call an elementary routine from 8.3.3, although the implementation is free to choose the optimal solution.

8.3.1 E2E Profile 1 routines

8.3.1.1 E2E_P01Protect

[SWS_E2E_00166]E2E_P01Protect

Service name:	E2E_P01Protect	
Syntax:	Std_ReturnType E2E_P01Protect (E2E_P01ConfigType* Config, E2E_P01SenderStateType* State, uint8* Data)	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Config	Pointer to static configuration.
Parameters (inout):	State	Pointer to port/data communication state.
	Data	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 E2E0047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID.	

]()

8.3.1.2 E2E_P01Check

[SWS_E2E_00158]E2E_P01Check

Service name:	E2E_P01Check	
Syntax:	Std_ReturnType E2E_P01Check (E2E_P01ConfigType* Config, E2E_P01ReceiverStateType* State, uint8* Data)	
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 E2E0047.
Description:	Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.	

]()

8.3.2 E2E Profile 2 routines

8.3.2.1 E2E_P02Protect

[SWS_E2E_00160]E2E_P02Protect

Service name:	E2E_P02Protect	
Syntax:	Std_ReturnType E2E_P02Protect (E2E_P02ConfigType* Config, E2E_P02SenderStateType* State, uint8* Data)	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Config	Pointer to static configuration.
	State	Pointer to port/data communication state.
Parameters (inout):	Data	Pointer to the data to be protected.
	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 E2E0047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.

]()

8.3.2.2 E2E_P02Check

[SWS_E2E_00161]E2E_P02Check

Service name:	E2E_P02Check	
Syntax:	Std_ReturnType E2E_P02Check(E2E_P02ConfigType* Config, E2E_P02ReceiverStateType* State, uint8* Data)	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Config	Pointer to static configuration.
	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 E2E0047.
Description:	Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.	

]()

8.3.3 Elementary protocol routines

The E2E Library provides various elementary functions enabling to build custom E2E profiles. First, it provides a couple of CRC routines, which are just wrappers above CRC library CRC8 functions, for computing CRC8 over multi-byte integers and for computing CRC8 over arrays of multi-byte integers. The CRC functions can be used by Software Components to calculate CRC on all the data elements in a complex data element. Secondly, the E2E Library provides functions for handling the counter and for handling error flags.

There are no known users/projects that use the functions specified in this section (8.3.3). Therefore, the functions specified in this section are considered as obsolete. In future, it is planned to either remove them all from E2E Library or to move some of them to CRC library. This will happen at earliest in R4.2.

Service ID[hex]:	0x07, 0x08, 0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0x00, or (2) 0x00 if it is the first run.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	CRC8 value calculated based on the CRC from previous iteration and over a primitive data element from the current iteration.
Description:	<p>InTypeMn: {u8, u16, u32}, which is the one corresponding to InType.</p> <p>Utility function for computing CRC over primitive data types transmitted with E2E Protocol, as in E2E Profile 1. The calculation is done in Least Significant Byte First, regardless of the architecture of the microcontroller, because this is the byte order in which data is transmitted over FlexRay, CAN and LIN. This function is provided also for uint8, which is redundant to the CRC function provided by the CRC library, but it makes the API more systematic.</p> <p>Relation to Crc_CalculateCRC8(): E2E_CRC8_*() may simply call Crc_CalculateCRC8() in a loop.</p> <p>The function uses SAE J1850 polynomial, but with 0x00 as start value and XOR value.</p>	

}]()

[SWS_E2E_00091]

Service ID[hex]	Function prototype
0x07	uint8 E2E_CRC8u8 (uint8 E2E_Data, uint8 E2E_StartValue)
0x08	uint8 E2E_CRC8u16 (uint16 E2E_Data, uint8 E2E_StartValue)
0x09	uint8 E2E_CRC8u32 (uint32 E2E_Data, uint8 E2E_StartValue)

}]()

8.3.3.2 E2E_CRC8<InTypeMn>Array

[SWS_E2E_00094]E2E_CRC8<InTypeMn>Array

Service name:	E2E_CRC8<InTypeMn>Array	
Syntax:	uint8	E2E_CRC8<InTypeMn>Array (uint16 Length, <InType>* Data, uint8 StartValue)
Service ID[hex]:	0x0A, 0x0B, 0x0C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Length	Length of array (data block) to be calculated in bytes.
	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0x00, or (2)

	0x00 if it is the first run.
Parameters (inout):	None
Parameters (out):	None
Return value:	uint8 CRC8 value calculated based on the CRC from previous iteration and over the array from the current iteration.
Description:	InTypeMn: {u8, u16, u32}, which is the one corresponding to InType. Utility function for calculating CRC over an array of primitive data types transmitted with E2E Protocol. The computation is done in Least Significant Byte First, regardless of the architecture of the microcontroller, because this is the byte order in which data is transmitted over FlexRay, CAN and LIN. This function is provided also for uint8, which is redundant to the CRC function provided by the CRC library, but it makes the API more systematic. Relation to Crc_CalculateCRC8(): E2E_CRC8_<InType>Array() may simply call Crc_CalculateCRC8() or E2E_CRC8_<InTypeMn>() in a loop. The function uses SAE J1850 polynomial, but with 0x00 as start value and XOR value.

]()

[SWS_E2E_00095]

Service ID[hex]	Function prototype
0x0A	uint8 E2E_CRC8u8Array (const uint8* E2E_DataPtr, uint32 E2E_ArrayLength, uint8 E2E_StartValue)
0x0B	uint8 E2E_CRC8u16Array (const uint16* E2E_DataPtr, uint32 E2E_ArrayLength, uint8 E2E_StartValue)
0x0C	uint8 E2E_CRC8u32Array (const uint32* E2E_DataPtr, uint32 E2E_ArrayLength, uint8 E2E_StartValue)

]()

8.3.3.3 E2E_CRC8H2F<InTypeMn>

Note: this function is introduced in E2E Library if CRC Library will support (in 4.0) the polynomial 0x2F.

[SWS_E2E_00096]E2E_CRC8H2F<InTypeMn>

Service name:	E2E_CRC8H2F<InTypeMn>	
Syntax:	uint8	E2E_CRC8H2F<InTypeMn>(<InType> Data, uint8 StartValue)
Service ID[hex]:	0x0D, 0x0E, 0x0F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0xFF, or (2) 0xFF if it is the first run.

Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	CRC8 value calculated based on the CRC from previous iteration and over a primitive data element from the current iteration, using CRC8 polynomial 0x2F.
Description:	<p>InTypeMn: {u8, u16, u32}, which is the one corresponding to InType.</p> <p>Utility function for calculating CRC over primitive data types transmitted with E2E Protocol. The computation is done in Least Significant Byte First, regardless of the architecture of the microcontroller, because this is the byte order in which data is transmitted over FlexRay, CAN and LIN. This function is provided also for uint8, which is redundant to the CRC function provided by the CRC library, but it makes the API more systematic. The function uses not the SAE polynomial, but 0x2F.</p>	

]()

[SWS_E2E_00276]

Service ID[hex]	Function prototype	
0x0D	uint8 E2E_CRC8H2Fu8	(uint8 E2E_Data, uint8 E2E_StartValue)
0x0E	uint8 E2E_CRC8H2Fu16	(uint16 E2E_Data, uint8 E2E_StartValue)
0x0F	uint8 E2E_CRC8H2Fu32	(uint32 E2E_Data, uint8 E2E_StartValue)

]()

8.3.3.4 E2E_CRC8H2F<InTypeMn>Array

For interoperability reasons, whenever possible, instead of using E2E_CRC8H2F<InTypeMn>Array(), one should use a corresponding E2E_CRC8<InTypeMn>Array().

[SWS_E2E_00097]E2E_CRC8H2F<InTypeMn>Array

Service name:	E2E_CRC8H2F<InTypeMn>Array	
Syntax:	uint8	E2E_CRC8H2F<InTypeMn>Array (uint16 Length, <InType>* Data, uint8 StartValue)
Service ID[hex]:	0x10, 0x11, 0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Length	Length of array (data block) to be calculated in bytes.
	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0xFF, or (2) 0xFF if it is the first run.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	CRC8 value calculated based on the CRC from previous iteration and over the array from the current iteration, using CRC8 polynomial 0x2F.

Description:	InTypeMn: {u8, u16, u32}, which is the one corresponding to InType. Utility function for calculating CRC over an array of primitive data types transmitted with E2E Protocol. The computation is done in Least Significant Byte First, regardless of the architecture of the microcontroller, because this is the byte order in which data is transmitted over FlexRay, CAN and LIN. This function is provided also for uint8, which is redundant to the CRC function provided by the CRC library, but it makes the API more systematic. The function uses not the SAE polynomial, but 0x2F.
---------------------	---

]()

[SWS_E2E_00098]

Service ID[hex]	Function prototype
0x10	uint8 E2E_CRC8H2Fu8Array (const uint8* E2E_DataPtr, uint32 E2E_ArrayLength, uint8 E2E_StartValue)
0x11	uint8 E2E_CRC8H2Fu16Array (const uint16* E2E_DataPtr, uint32 E2E_ArrayLength , uint8 E2E_StartValue)
0x12	uint8 E2E_CRC8H2Fu32Array (const uint32* E2E_DataPtr, uint32 E2E_ArrayLength , uint8 E2E_StartValue)

]()

8.3.3.5 E2E_UpdateCounter

[SWS_E2E_00099]E2E_UpdateCounter

Service name:	E2E_UpdateCounter	
Syntax:	uint8	E2E_UpdateCounter (uint8 Counter)
Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Counter	Counter value, to be incremented.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	Incremented counter value.
Description:	Increments the counter provided by the parameter, and returns it by return value. The routine is very simple: return value = (Counter++) % 15. This means that the counter takes values 0..14 and the next value after 14 is 0. Value 15 (i.e. 0xF) is reserved as invalid value.	

]()

8.3.4 Auxiliary Functions

8.3.4.1 E2E_GetVersionInfo

[SWS_E2E_00032]E2E_GetVersionInfo

Service name:	E2E_GetVersionInfo
Syntax:	void E2E_GetVersionInfo (Std_VersionInfoType* VersionInfo)
Service ID[hex]:	0x14
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	VersionInfo Pointer to where to store the version information of this module.
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Returns the version information of this module.

](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]()

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.]()

8.6.1 Mandatory Interfaces

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

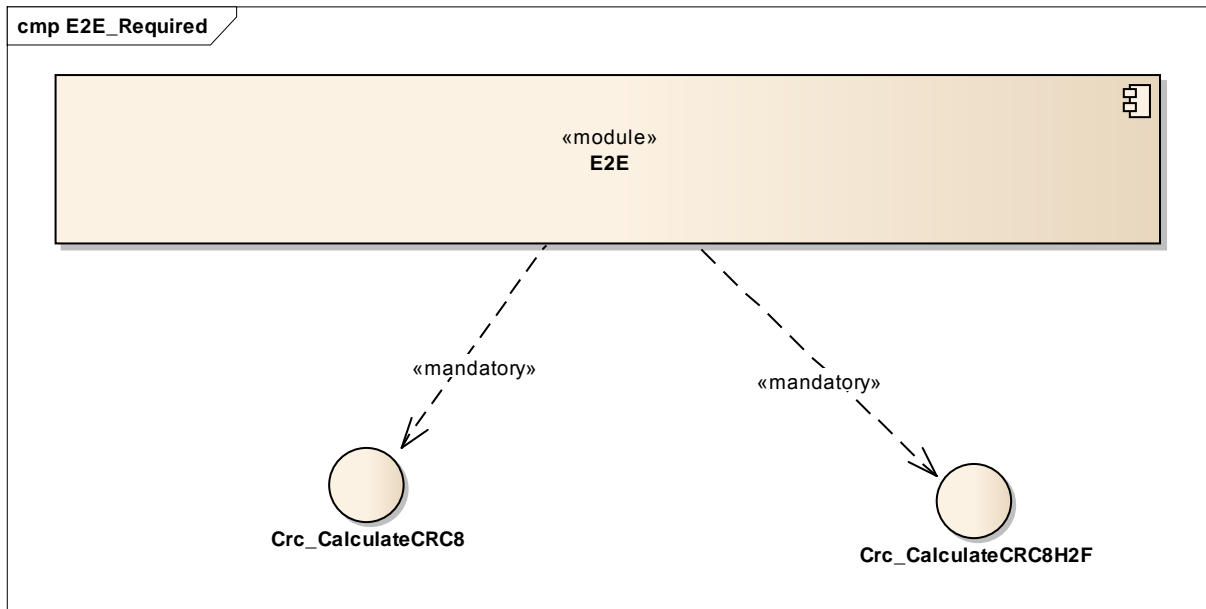


Figure 8-4: Expected mandatory interfaces by E2E library

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 PXXSenderStateType, separately for each Data to be protected.]()

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

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_PXXSenderStateType for each Data, with the configured following values: Counter = 0.]()

[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.

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.

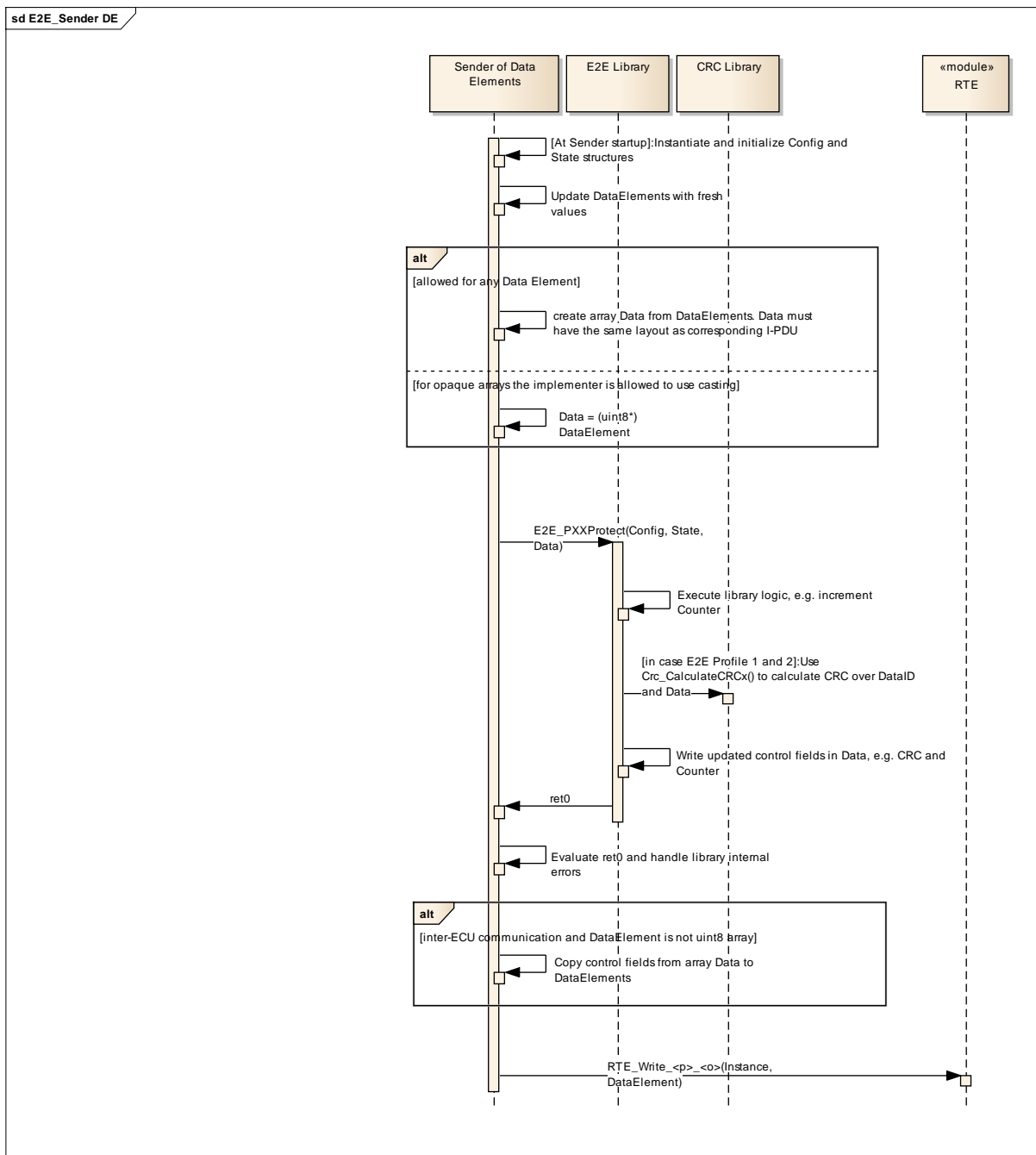


Figure 9-1: Sender of data elements

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 the equal to the content of the serialized representation of corresponding signal group in an I-PDU.]()

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.]()

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.

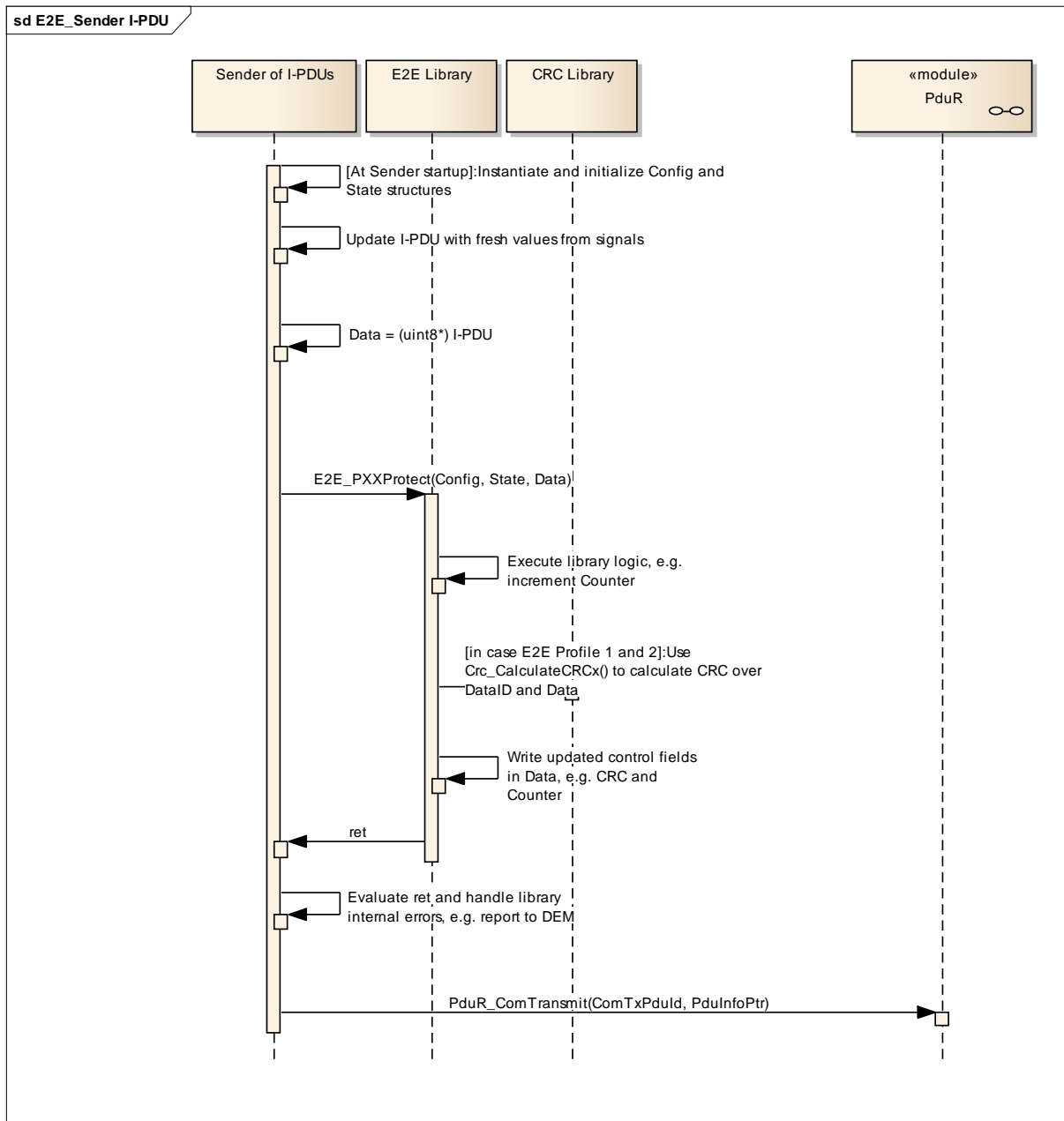


Figure 9-2: Sender at signal group level

9.2 Receiver

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

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 = 4 (i.e. tolerance interval for maximum counter differences)
 Config → SyncCounterInit = 3 (i.e. duration of counter continuity check)
 Timeout interval checked by SWC = 8 transmission cycles

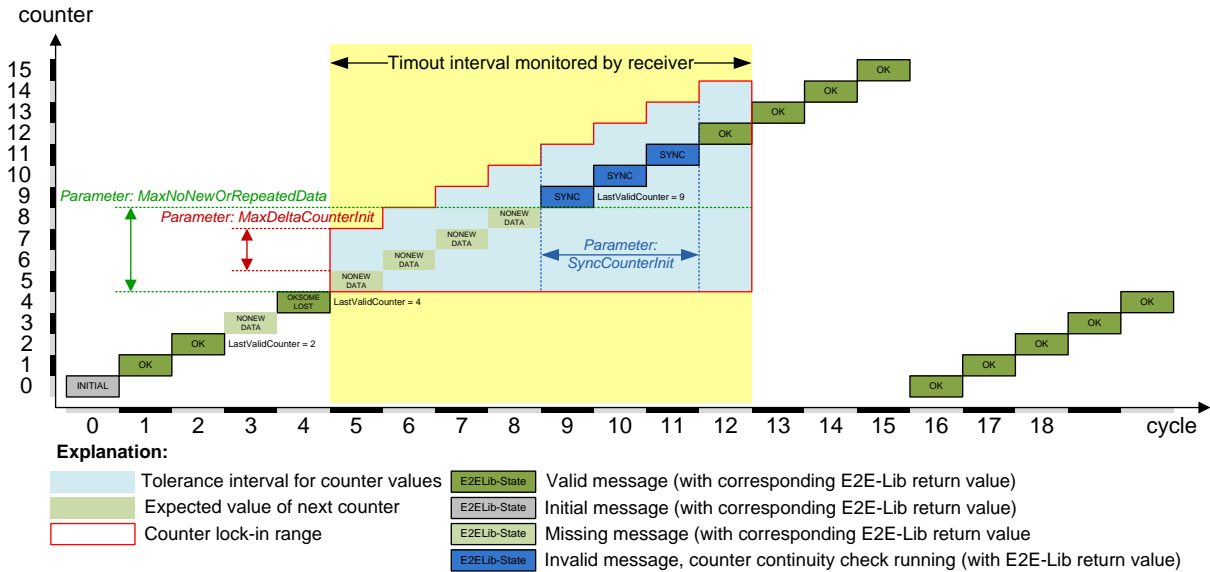


Figure 9-3: Configuration parameters of the E2E_PxxCheck() function and their effects

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

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

[UC_E2E_00208] [During its initialization, the Receiver shall initialize the E2E_PXXReceiverStateType with the following values:

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

[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().|()

The Functions E2E_PXXCheck() return the results of verification, by means of parameter State. Within the State (structure E2E_PXXReceiverStateType), 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.|()

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.

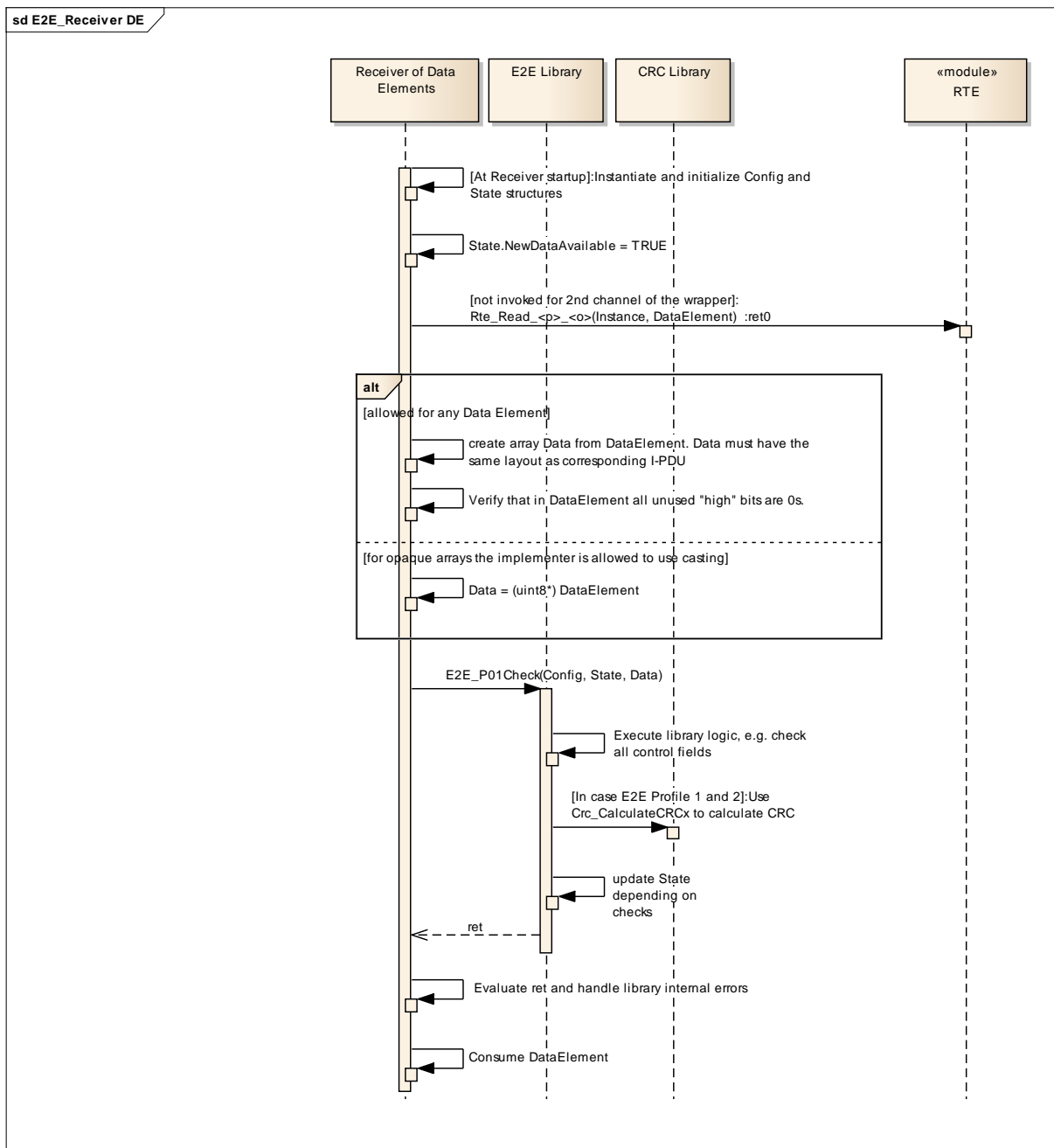


Figure 9-4: Receiver at data element level

[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.]()

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.]()

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.

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

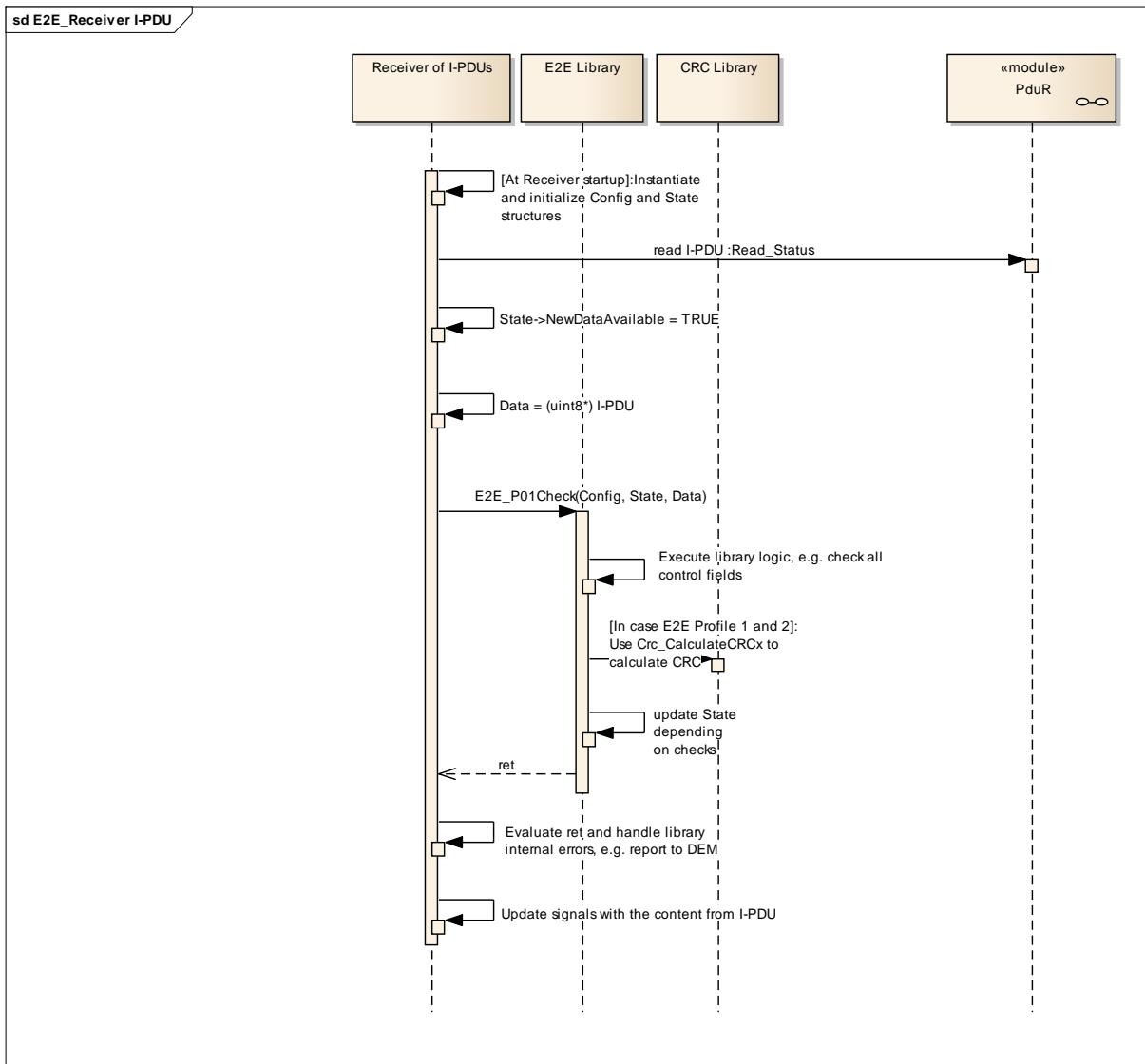


Figure 9-5: Receiver at signal group level

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.]()

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.]()

11.3 Maximal lengths of Data, communication buses

The length of the message and the achieved hamming distance for a given CRC are related. To ensure the required diagnostic coverage the maximum length of data elements protected by a CRC needs to be selected appropriately.

The E2E profiles are intended to protect inter-ECU communication with lengths as listed in the table below (see Figure 11-1).

E2E Profile	Max applicable length including control fields for inter-ECU communication
E2E Profile 1	32
E2E Profile 2	32

Figure 11-1: Maximum lengths

The Hamming Distance is 2, up to the given lengths. Due to 8 bit CRC, the burst error detection is up to 8 bits.

[UC_E2E_00051][In case of inter-ECU communication over FlexRay or CAN, the length of the complete Data (including application data, CRC and counter) protected by E2E Profile 1 or E2E Profile 2 should not exceed 32 bytes.]()

This requirement only contains a reasonable maximum length evaluated during the design of the E2E profiles. The responsibility to ensure the adequacy of the implemented E2E protection using E2E Library for a particular system remains by the user.

[UC_E2E_00061][In case of CAN or LIN the length of the complete data element (including application data, CRC and counter) protected by E2E Profile 1 should not exceed 8 bytes.]()

The requirements [UC_E2E_00051](#) and [UC_E2E_00061](#) only contain a reasonable maximum length evaluated during the design of the E2E profiles.

[UC_E2E_00236][When using E2E Library, the designer of the functional or technical safety concept of a particular system using E2E Library shall evaluate the maximum permitted length of the protected Data in that system, to ensure an appropriate error detection capability.]()

Thus, the specific maximum lengths for a particular system may be shorter (or maybe in some rare cases even longer) than the recommended maximum applicable lengths defined for the E2E Profiles.

[UC_E2E_00170][When designing the functional or technical safety concept of a particular system any user of E2E Library shall ensure that the transmission of one undetected erroneous data element in a sequence of data elements between sender and receiver will not directly lead to the violation of a safety goal of this system.

In other words, SW-C shall be able to tolerate the reception of one erroneous data element, which error was not detected by the E2E library. What is *not* required is that an SW-C tolerates two consecutive undetected erroneous data elements, because it is enough unlikely that two consecutive Data are wrong AND that for both Data the error remains undetected by the E2E library.]()

When using LIN as the underlying communication network the residual error rate on protocol level is several orders of magnitude higher (compared to FlexRay and CAN) for the same bit error rate on the bus. The LIN checksum compared to the protocol CRC of FlexRay (CRC-24) and CAN (CRC-15) has different properties (e.g. hamming distance) resulting in a higher number of undetected errors coming from the bus (e.g. due to EMV). In order to achieve a maximum allowed residual error rate

on application level, different error detection capabilities of the application CRC may be necessary, depending on the strength of the protection on the bus protocol level.

11.4 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 12.

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 `ISignalIPdu` 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.

11.5 Configuration constraints on Data IDs

11.5.1 Data IDs

To be able to verify the identity of the data elements or signal groups, none of two are allowed to have the same Data ID (E2E Profile 1) or same `DataIDList[]` (E2E Profile 2) within one system of communicating ECUs.

It is recommended that the value of the Data ID be assigned by a central authority rather than by the developer of the software-component. The Data IDs are defined in Software Component Template, and then realized in E2E_PXXConfig structures.

[UC_E2E_00071] [Any user of E2E Library shall ensure that within one implementation of a communication network every safety-related data element, protected by E2E Library, has a unique Data ID (see Profile 1) or a unique DataIDList[] (see Profile 2).]()

[UC_E2E_00237] [Any user of E2E Library shall ensure, that within one implementation of a communication network every safety-related Data, protected by E2E Library, has a unique Data ID (see Profile 1) or a unique DataIDList[] (see Profile 2).]()

Note: For Profile 1 requirement ([UC E2E 00071](#)) may not be sufficient in some cases, because Data ID is longer than CRC, which results with additional requirements [UC E2E 00072](#) and [UC E2E 00073](#). In Case of Profile 1 the ID can be encoded in CRC by double Data ID configuration (both bytes of Data ID are included in CRC every time), or in alternating Data ID configuration (high byte or low byte of Data ID are put in CRC alternatively, depending of parity of Counter), there are different additional requirements/constraints described in the sections below.

11.5.2 Double Data ID configuration of E2E Profile 1

In E2E Profile 1, the CRC is 8 bits, whereas Data ID is 16bits. In the double Data ID configuration (both bytes of Data ID are included in CRC every time), like it is in the E2E variant 1A, all 16 bits are always included in the CRC calculation. In consequence, two different 16 bit Data IDs DI1 and DI2 of data elements DE1 and DE2 may have the same 8 bit CRC value. Now, a possible failure mode is for example that a gateway incorrectly routes a safety-related signal DE1 to the receiver of DE2. The receiver of DE2 receives DE1, but because the DI1 and DI2 are identical, the receiver might accept the message (this assumes that by accident the counter was also correct and that possibly data length was the same for DE1 and DE2).

To resolve this, there are additional requirements limiting the usage of ID space. Data elements with ASIL B and above shall have unique CRC over their Data ID, and signals having ASIL A requirements shall have a unique CRC over their Data IDs for a given data element/signal length.

[UC_E2E_00072] [Any user of Profile 1 in Double Data ID configuration shall ensure that assuming two data elements DE1 and DE2 on the same system (vehicle): for any data element DE1 having ASIL B, ASIL C or ASIL D requirements with Data ID DI1, there shall not exist any other data element DE2 (of any ASIL) with Data ID DI2, where:

```
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI1),highbyte (DI1)} )
=
```

```
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI2),highbyte(DI2)} ) |()
```

The above requirement limits the usage of Data IDs of data having ASIL B, C, D to 255 distinct values in a given ECU, but gives the flexibility to define the Data IDs within the 16-bit naming space.

For data elements having ASIL A requirements, the requirement is weaker – it requires that there are no CRC collisions for the ASIL A signals of the same length:

[UC_E2E_00073] [Any user of Profile 1 in Double Data ID configuration shall ensure, that assuming two data elements DE1 and DE2, on the same system (vehicle): for any data element DE1 having ASIL A requirements with Data ID DI1, there shall not exist any other data element DE2 (having ASIL A requirements) with Data ID DI2 and of the same length as DE1, where

```
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI1),highbyte(DI1)} )  
=
```

```
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI2),highbyte(DI2)} ) |()
```

The above two requirements [UC_E2E_00072](#) and [UC_E2E_00073](#) assume that DE1 and DE2 are on the same system. If DE1 and DE2 are exclusive (i.e. either DE1 or DE2 are used, but never both together in the same system / vehicle configuration, e.g. DI is available in coupe configuration and DI2 in station wagon configuration), then CRC(DI1) = CRC(DI2) is allowed.

11.5.3 Alternating Data ID configuration of E2E Profile 1

In the alternating Data ID configuration, either high byte or low byte of Data ID is put in CRC alternatively, depending of parity of Counter. In this configuration, two consecutive Data are needed to verify the data identity. This is not about the reliability of the checksum or software, but really the algorithm constraint, as on every single Data only a single byte of the Data ID is transmitted and therefore it requires two consecutive receptions to verify the Data ID of received Data.

11.5.4 Nibble configuration of E2E Profile 1

In the nibble Data ID configuration of E2E Profile 1, the low byte is not transmitted, but included in the CRC. Because the low byte has the length of 8 bits, it is the same as the CRC. Therefore, if two Data IDs are different in the low byte, this results with a different CRC over the Data ID low byte.

[UC_E2E_00308] [

1. Any user of Profile 1 in Nibble Data ID configuration shall ensure that:the high nibble of high byte of Data ID is equal to 0
2. the low nibble of high byte of Data ID is within the range 0x1..0xE (to avoid collisions with other E2E Profile 1 configurations that have 0x0 on this nibble, and to exclude the invalid value 0xF).

3. The low byte of Data ID is different to low byte of any Data ID present in the same bus that uses E2E Profile in Double Data ID configuration.]()

[UC_E2E_00317] [When using E2E Profiles 1A and 1C in one bus/system, the following shall be respected:

1. 1A data shall use IDs that are < 256 (this means high byte shall be always = 0)
2. 1C data shall use IDs that are ≥ 256 (this means high byte is always $\neq 0$) and $< 4^096$ (0x10'00 - it means they fit to 12 bits).
3. Any low byte of 1C data id shall be different to any low byte of 1A data ID.]()

Thanks to the Data ID distribution according to the above requirement, addressing errors can be detected: in particular, it can be detected when 1C message arrives to 1A destination. If 1C message receives to a 1A destination, then the CRC check will pass if low byte of the sent 1C message equals to the expected 1A address - and this is excluded by the above requirement.

Example: 1A may use addresses 0 to 199, while 1C may use addresses where low byte is 200 to 255 and high byte is between 1 and 15. This allows to use additional $(256-200)*15 = 840$ Data IDs.

11.6 Building custom E2E protocols

E2E Library offers elementary functions (e.g. for handling CRC and alive counters), from which non-standard protocols can be built. It is within the responsibility of the integrator/application developer to come up with a correct protocol. A custom E2E protocol can be built as an SW-C or as a custom (non-standard) BSW library.

[UC_E2E_00259] [Any developer of a custom-built E2E Profile using elementary mechanisms provided by E2E Library shall ensure that this custom built E2E Profile is adequate for safety-related communications within the automotive domain.]()

A list of CRC routines is provided by E2E Library. CRC should be calculated on the bytes and bits of the data elements in the same order as in which it is transmitted on hardware bus. To be able to do this, the microcontroller Endianness and the used bus must be known. Once it is known, the corresponding E2E Library CRC routines should be used.

11.7 I-PDU Layout

This chapter provides some requirements and recommendations on how safety-related I-PDUs shall or should be defined. These recommendations can be also extended to non-safety-related I-PDUs.

11.7.1 Alignment of signals to byte limits

This chapter provides some requirements and recommendation on how safety-related data structures (e.g. signal-groups or I-PDUs) shall or can be defined. They could also be extended to non-safety-related data structures if found adequate.

[UC_E2E_00062][Signals that have length < 8 bits should be allocated to one byte of an I-PDU, i.e. they should not span over two bytes.]()

[UC_E2E_00063][Signals that have length >= 8 bits should start or finish at the byte limit of an I-PDU.]()

The previous recommendations cause that signals of type uint8, uint16 and uint32 fit exactly to respectively one, two or four byte(s) of an I-PDU.

These recommendations also cause that for uint8, uint16 and uint32, the bit offsets are a multiple of 8.

The figure is an example of signals (CRC, Alive and Sig1) that are not aligned to I-PDU byte limits:



Figure 11-2: Example for alignment not following recommendations

11.7.2 Unused bits

It can happen that some bits in a protected data structure (e.g. signal group or I-PDU transmitted over a communication bus) are unused. In such a case, the sender does not send signals represented by these bits, and the receiver does not expect to receive signals represented by these bits. In order to have a systematically defined data structure and sender-receiver behavior, the unused bits are set to the defined default value before calculation of the CRC.

[UC_E2E_00173][Any caller of the E2E library at the sender side shall fill all unused areas in a signal group (i.e. bits for which no explicitly defined signals exist within the signal group) to a default value configured for the I-PDU associated to the signal group (system template parameter ISignalIPdu.unusedBitPattern).]()

The attribute unusedBitPattern is actually an 8-bit byte pattern. It can take any value from 0x00 to 0xFF. Often 0xFF is used.

If unused bits are replaced in a later point by a signal, then all receivers of that signal group that use the E2E Protection Wrapper need to be updated.

This means that replacing unused bits with a signal instead requires an update of all receiver ECUs that use E2E Protection Wrapper approach. As an alternative, one may define dummy signals (and corresponding data elements) for all unused areas within a signal group.

11.7.3 Byte order (Endianness)

For each signal that is longer than 1 byte (e.g. uint16, uint32), the bytes of the signal need to be placed in the I-PDU in a sequence. There are two ways to do it:

1. start with the *least* significant byte first – the significance of the byte *increases* with the increasing byte significance. This is called little Endian (i.e. little end first),
2. start with the *most* significant byte first - the significance of the byte *decreases* with the increasing byte significance. This is called big Endian (i.e. big end first).

For primitive data elements, RTE simply maps application data elements to COM signals, which means that RTE just copies/maps one variable to another one, both having the same data type.

COM in contrary is responsible for copying each signal into/from an I-PDU (i.e. for serialization of set of variables into an array). An I-PDU is transmitted over a network without any alteration. Before placing a signal in an I-PDU, COM can, if needed, change the byte Endianness the value:

1. Sender COM converts the byte Endianness of the signals (if configured/needed),
2. Sender COM copies the converted signal on I-PDU (serializes the signal), while copying only used bits from the signals,
3. Sender COM delivers unaltered I-PDU to receiver COM (an I-PDU is just a byte array unaltered by lower layers of the network stack),
4. Receiver COM converts the Endianness of the signals in the received I-PDU (if configured). It may also do the sign extension (if configured),
5. Receiver COM returns the converted signals.

Both sender and receiver COM can do byte Endianness conversion. Moreover, only receiver COM can do sign extension.

To achieve high level of interoperability, the automotive networks recommend a particular byte order, which is as follows:

Network	Byte order
FlexRay	Little Endian
CAN	Little Endian
LIN	Little Endian
Byteflight (not supported by AUTOSAR)	Big Endian

MOST (not supported by AUTOSAR)	Big Endian
---------------------------------	------------

Table 11-1: Networks and their byte order

The networks targeted by E2E, which are FlexRay, CAN and LIN are Little Endian, which results with the following requirement:

[UC_E2E_00055][Any user of E2E Profile 1 and 2 designing a SW-C shall place multibyte data in the data elements in Little Endian order.]()

AUTOSAR has two categories of data types: “normal” ones, which Endianness is/can be converted, and “opaque”, for which COM does not do any conversions. An opaque uint8 array is mapped one-to-one to an I-PDU. This results with the following requirements:

The below requirement simply says that either the signal is on both sides opaque, or on both sides non-opaque:

[UC_E2E_00057] [Any user of E2E Library shall ensure that a signal/data element is either opaque or non-opaque on both sides (i.e. the sender and the receiver side).

For example, a signal/data element as non-opaque on sender side and opaque on receiver side or vice versa are not allowed.]()

The below requirement states that if opaque types are used, then conversions (if needed) is done by software components.

[UC_E2E_00056] [Any user of E2E Library shall ensure that if an opaque data type is used as a signal/data element by a SW-C, then the signal/data element is in Little Endian order and the sender/receiver SW-C is responsible for converting the data element from/to the Little Endian order to its own byte order.]()

11.7.4 Bit order

There are two typical ways to store the bits of a byte:

1. most significant bit first (MSB first)
2. or least significant bit first (LSB first).

At the level of software, the microcontroller bit order is not visible. For example, a software module, accessing a bit 3 (of value 2^3) does not care or know if the bit is 3^{rd} stored by microcontroller as 3^{rd} from “left” (for LSB first) or 3^{rd} from “right” (for MSB first). Another important example is the CRC calculation: a CRC8 operates over values (e.g. looks up a value from lookup table at a given index). A function `CRC8(val1, prev): val2` returns always the same value, regardless of the microcontroller bit order. Well the values `val1`, `val2`, `prev` are the same in both cases, but they are stored inversely depending if it is MSB first or LSB first.

However, the bit order is in contrary relevant if a value is transmitted over a network, because the bit order determines in which network bit order determines in which order the bits are transmitted on the network. When data is copied from microcontroller memory to network hardware, the bit order takes place if microcontroller bit order is different from the network bit order.

Each network transmits a given byte in a particular bit order:

Network	Bit order
FlexRay	MSB first
CAN	MSB first
LIN	LSB first
Byteflight (not supported by AUTOSAR up to Release 4.0)	MSB first
MOST (not supported by AUTOSAR up to Release 4.0)	MSB first

Table 11-2: Networks and their bit order

To summarize above table, all listed networks apart from LIN are MSB first.

The bit order of the microcontroller is independent from the bit order of the network, but in all cases (combinations of different bit endianness of network sender and receiver microcontrollers) there is no impact on E2E or user of E2E due to bit order.

11.8 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.

11.8.1 Communication model for SW-C level protection

AUTOSAR RTE supports different communication models, like client-server, sender-receiver, mode switch etc. However, only the sender-receiver model is supported if the protection is realized at the level of data elements.

[UC_E2E_00087][In case the E2E Library is used to protect data elements, then the user of E2E Library shall use the Sender-receiver communication model for safety-related communication.]()

11.8.2 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.]()

11.8.3 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”)

As explained in 7.3.5, 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")]()

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 (e.g. memory partitioning).

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 to protect safety-related data elements exchanged between SW-Cs by means of E2E Protection Wrapper (Chapter 12.1). Second, E2E Library can be also used to protect safety-related data elements at the level of signal groups, by E2E COM callouts (Chapter 12.2).

It is also possible to have mixed scenarios:

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 [12] and Software Component Template [11]. 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 ISignalPdu metaclass (representing the I-PDU).

[UC_E2E_00271][A given I-PDU shall not be at the same protected by means of COM E2E callouts (through association with ISignalPdu) and by means of E2E Protection Wrapper (through association with E2E Protection Wrapper.]()

12.1 E2E Protection Wrapper

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

¹ The 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.

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 [12] and some of them are in Software Component Template [11].]()

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

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.]()

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.

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):

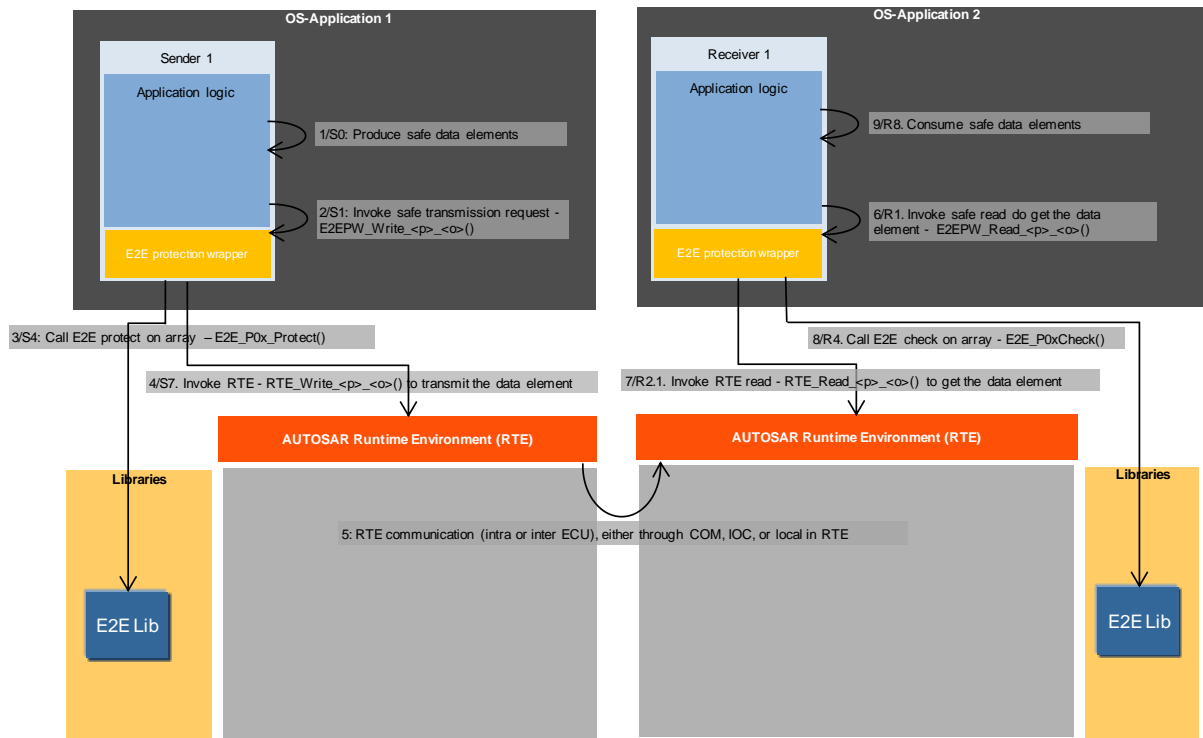


Figure 12-1: Example E2E Protection Wrapper - overall flow

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.

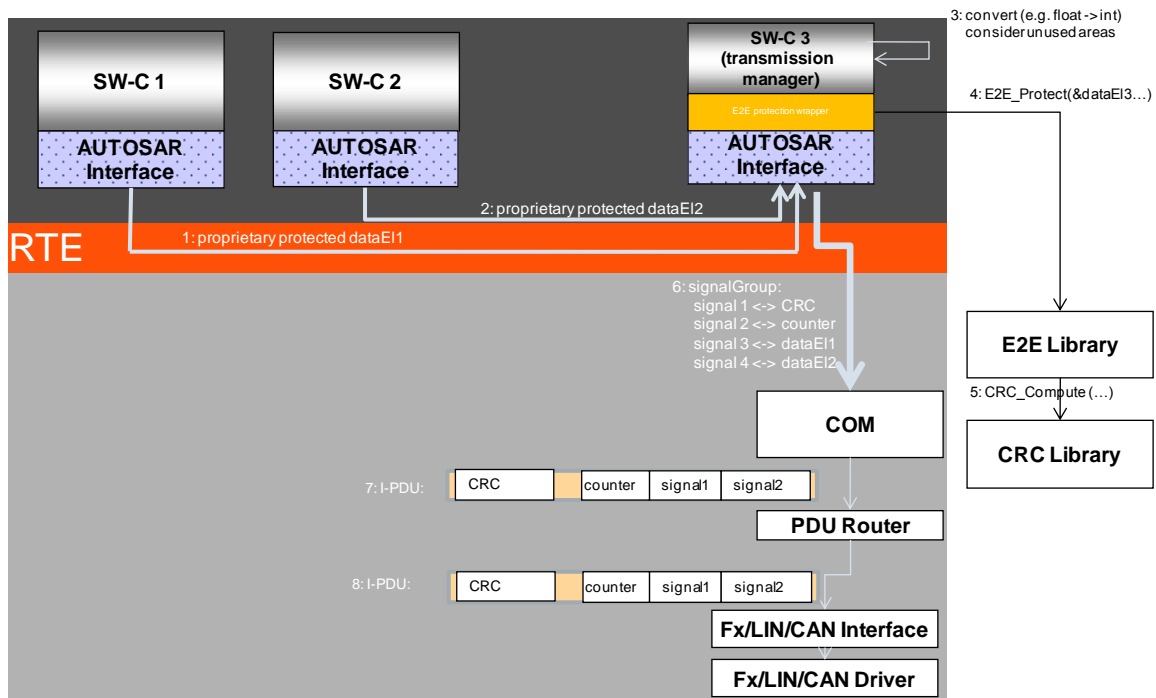


Figure 12-2: Example Transmission Manager – sender ECU

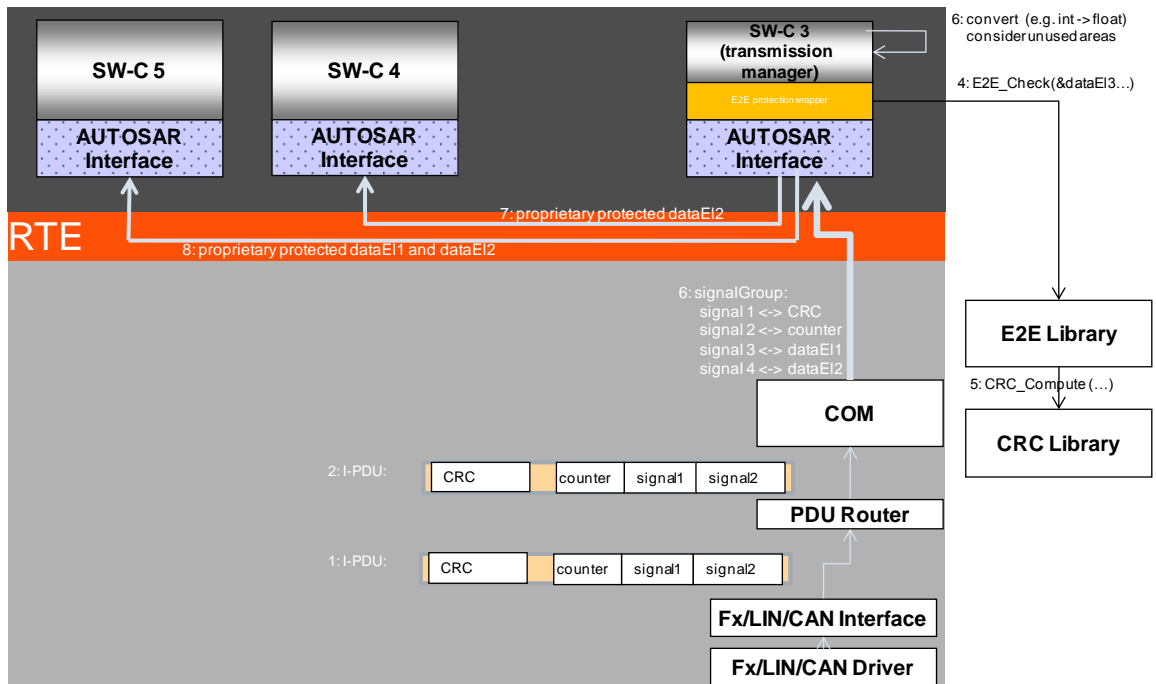


Figure 12-3: Example Transmission Manager – receiver ECU

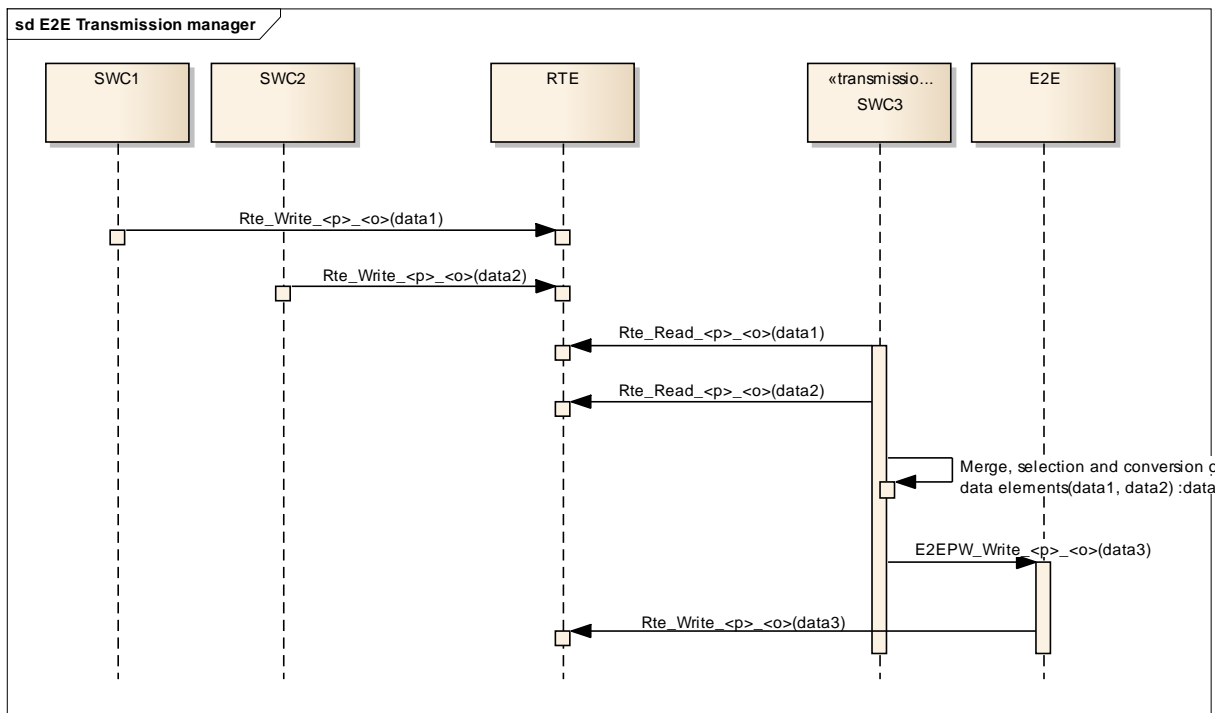


Figure 12-4: Example Transmission Manager – sender ECU sequence

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.]()

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.

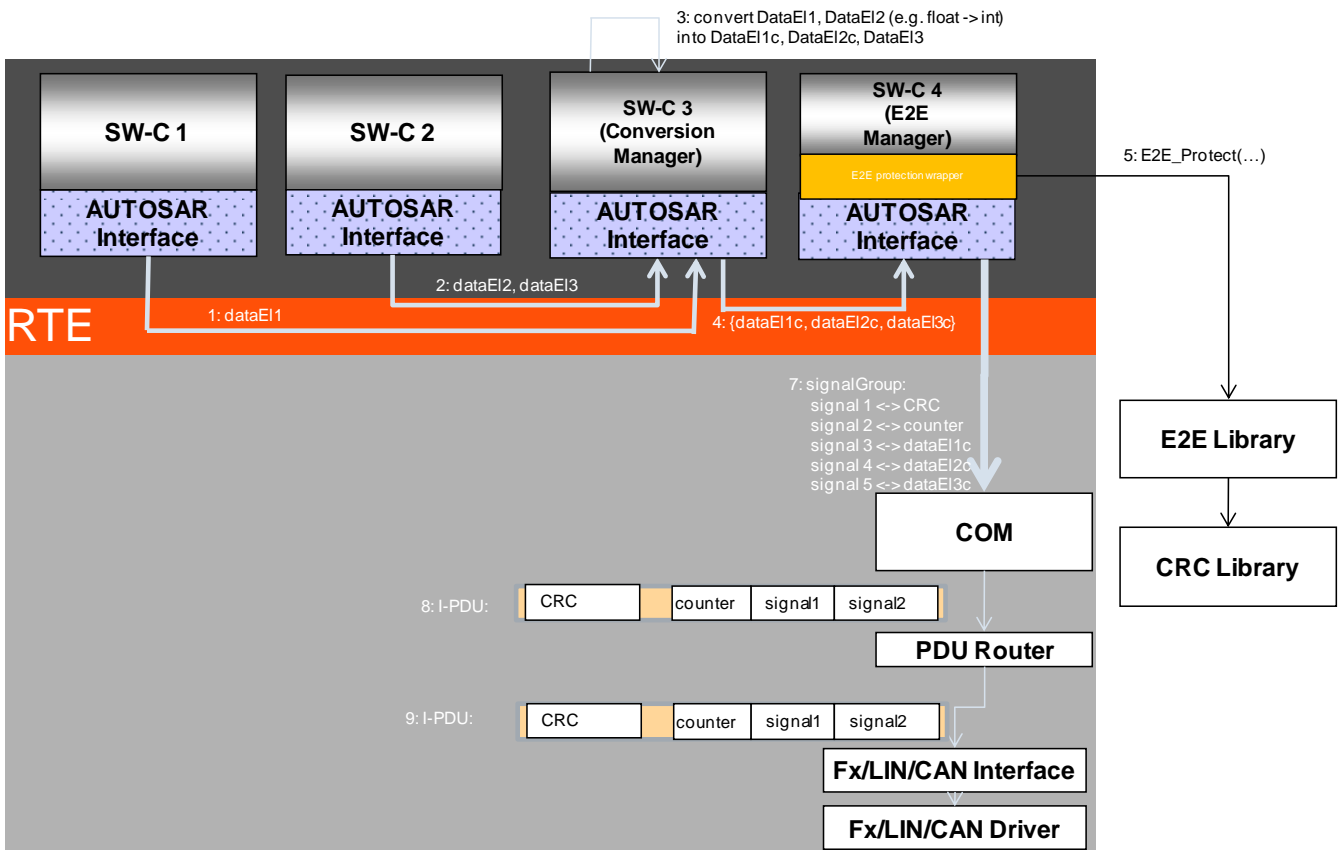


Figure 12-5: E2E Manager and Conversion Manager – sender ECU

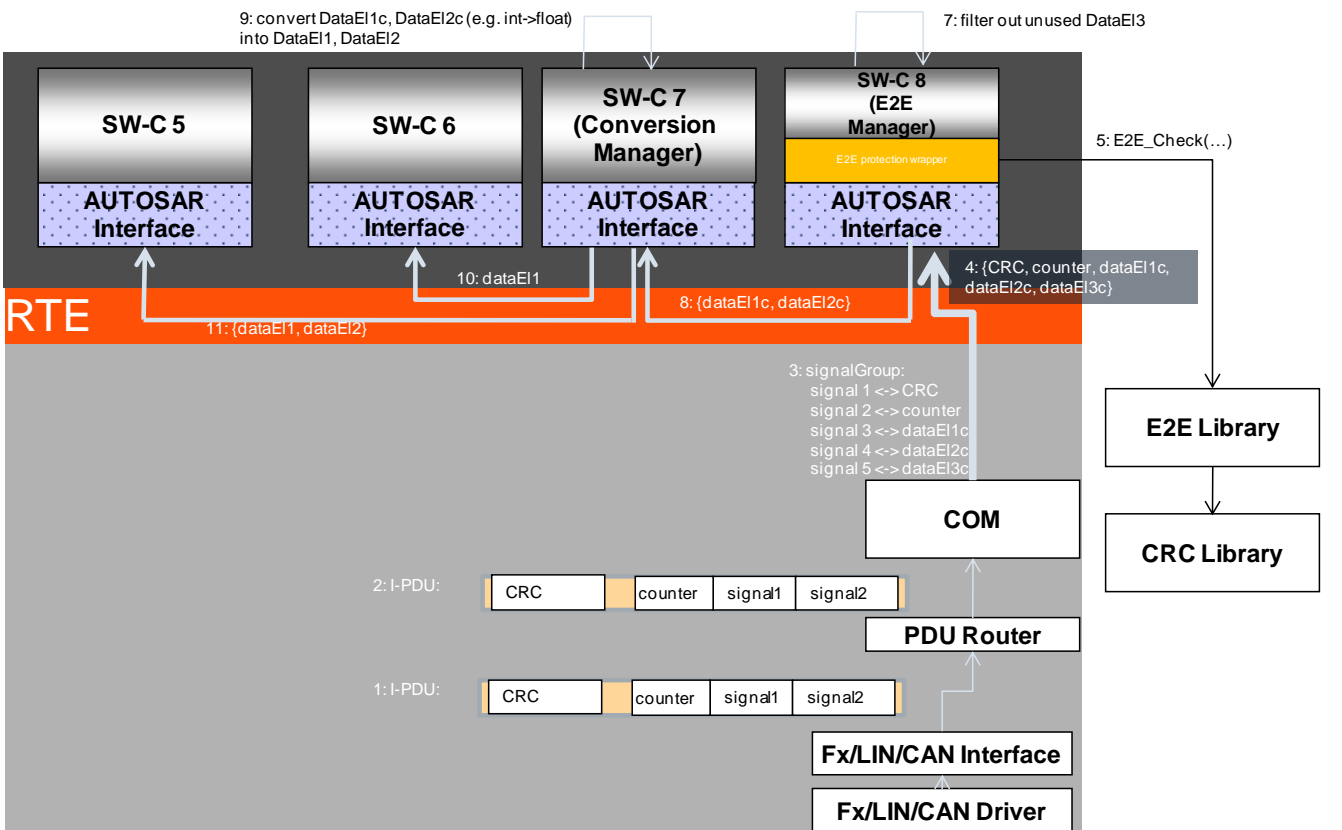


Figure 12-6: E2E Manager and Conversion Manager – receiver ECU

In the above example, the SW-Cs of sender ECU generate three data elements (dataE11, dataE12 and dataE13) 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:

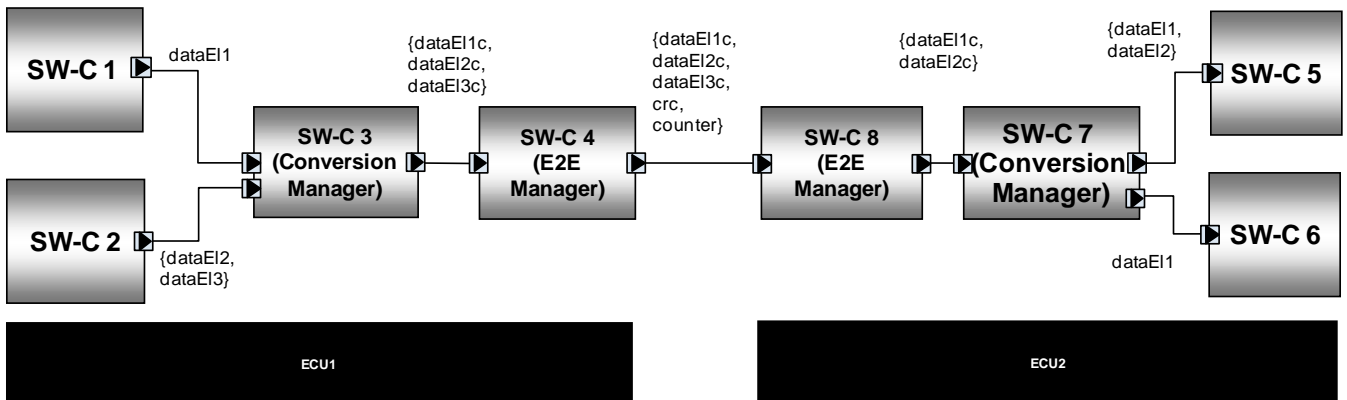


Figure 12-7: E2E Manager and Conversion Manager - system configuration

The E2E protection wrapper of E2E manager can be automatically generated, as described in 12.1.8.

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.]()

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).]()

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.]()

[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.]()

12.1.4 File structure

The figure below shows the required file structure of E2E Protection Wrapper.

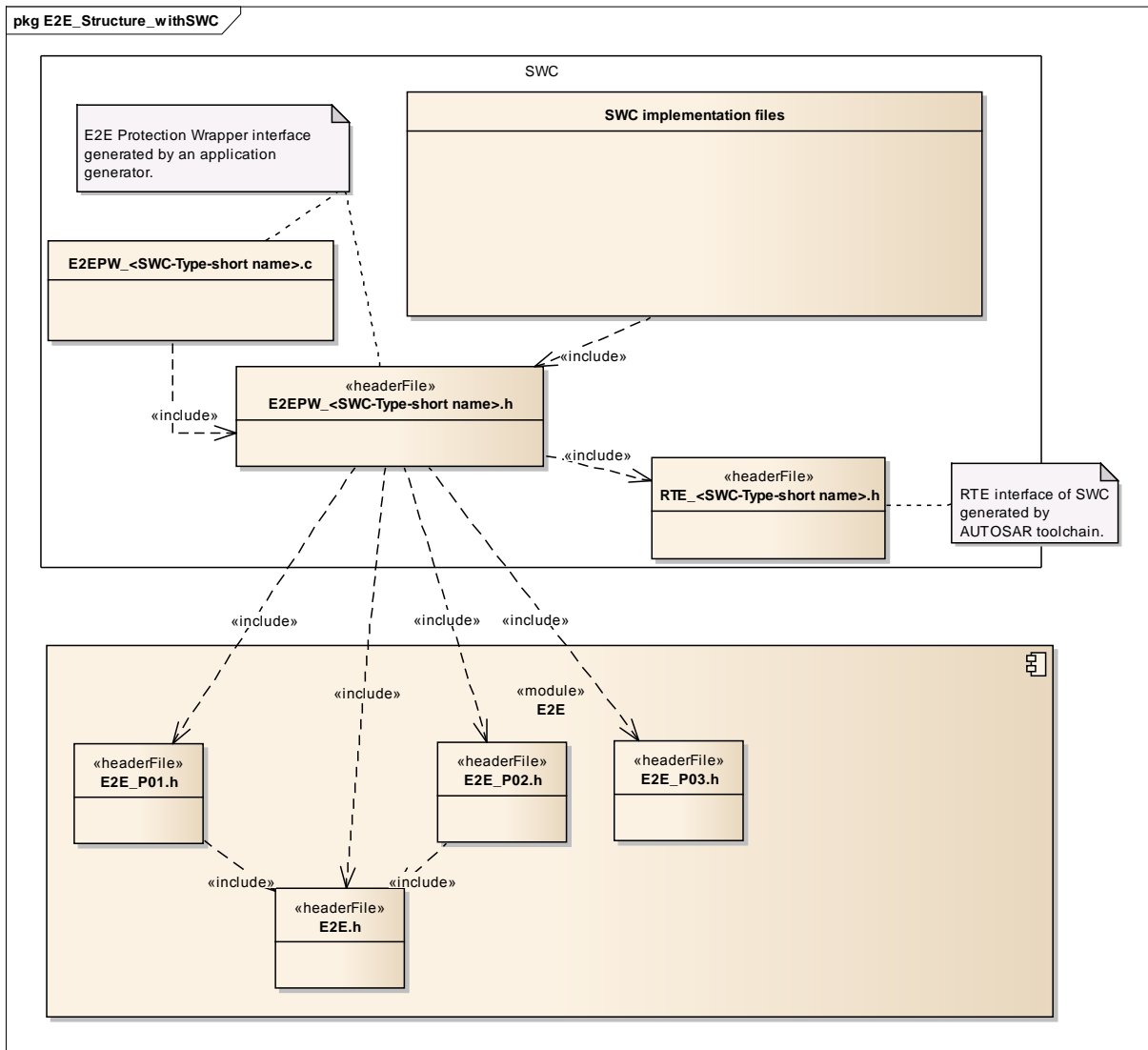


Figure 12-8: E2E File dependencies

[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.]()

[UC_E2E_00240][E2EPW_<SWC-Type-short name>.c shall include E2EPW_<SWC-Type-short name>.h.]()

[UC_E2E_00241][E2EPW_<SWC-Type-short name>.h shall include used header files from E2E Library (used E2E_PXX.h files) and shall include Rte_<SWC-Type-short name>.h.]()

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

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

[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.]()

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.]()

[UC_E2E_00289][If the E2EProtection is done in the E2E Wrapper then both EndToEndProtectionISignallPdu and EndToEndProtectionVariablePrototype shall be defined.]()

Most of the settings are defined under Software Component Template [11].

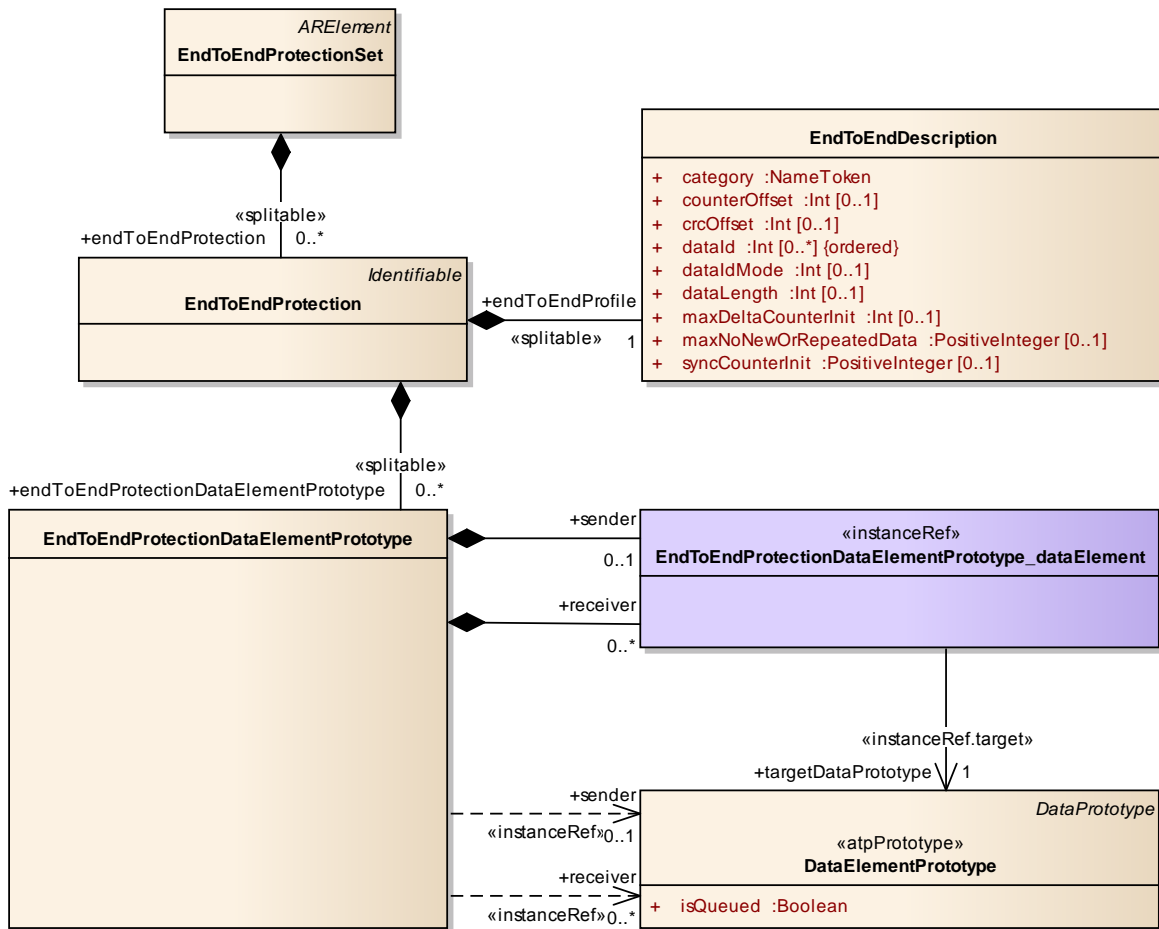


Figure 12-9: Release R3.2.x: E2E Protection Wrapper configuration (hardcopy from DOC_EndtoEndProtection)

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 [12]:

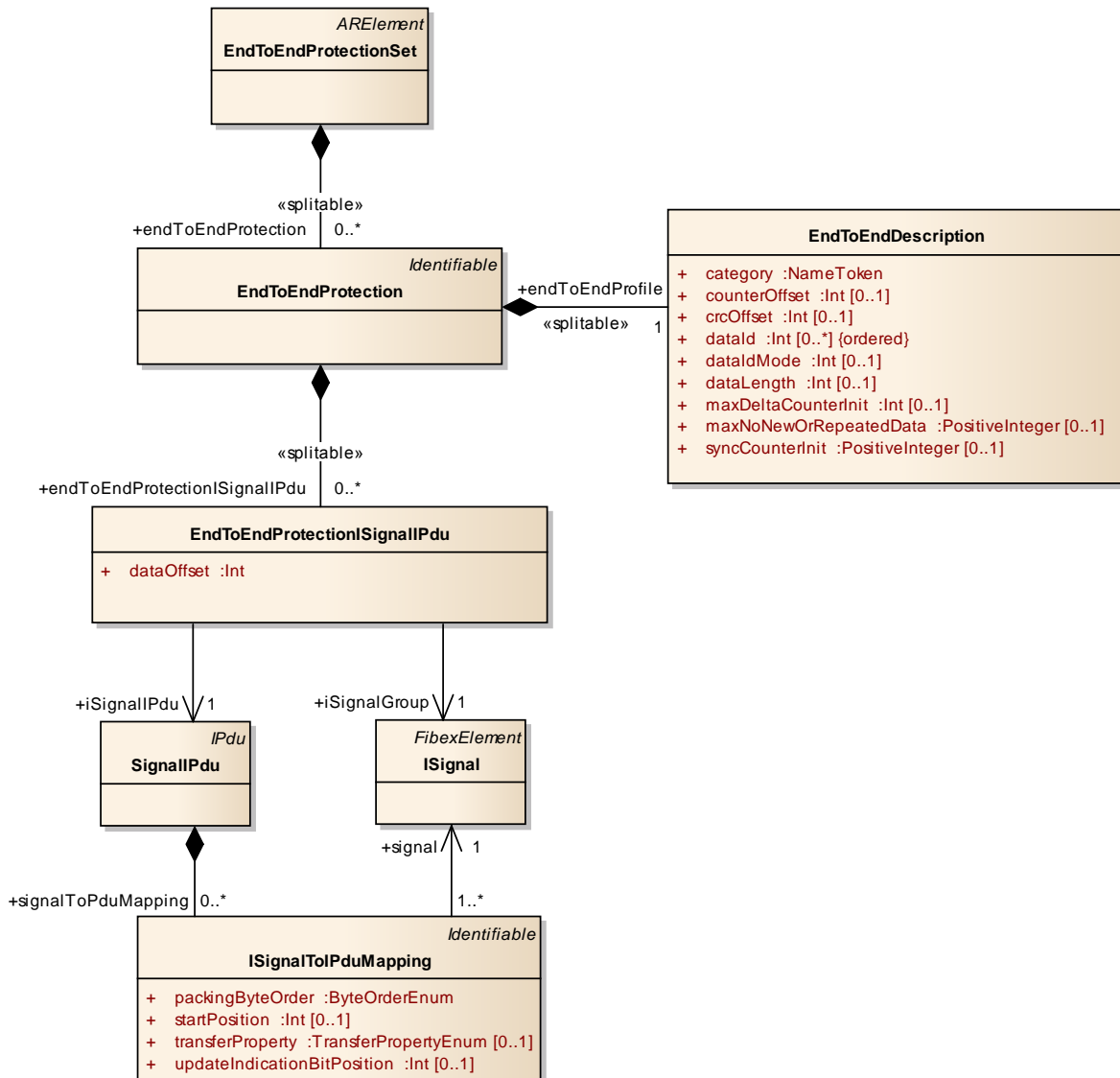


Figure 12-11: Release R3.2.x: E2E Protection Wrapper configuration (hardcopy from DOC_PduEndToEndProtection)

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 ISignalIPdu.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.

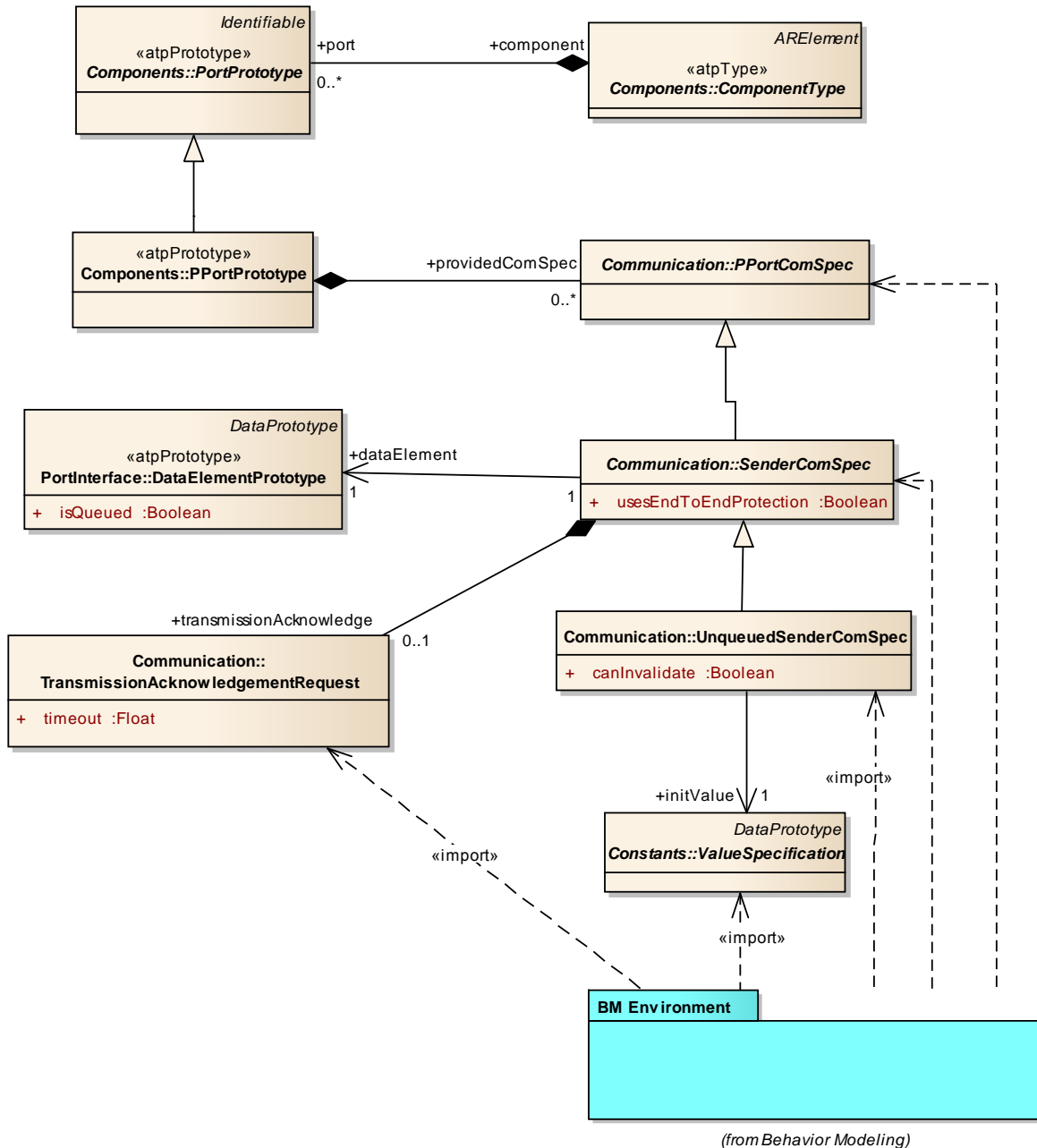


Figure 12-13: Release R3.2.x: SenderComSpec (hardcopy from BM ComSpec SenderComSpec)

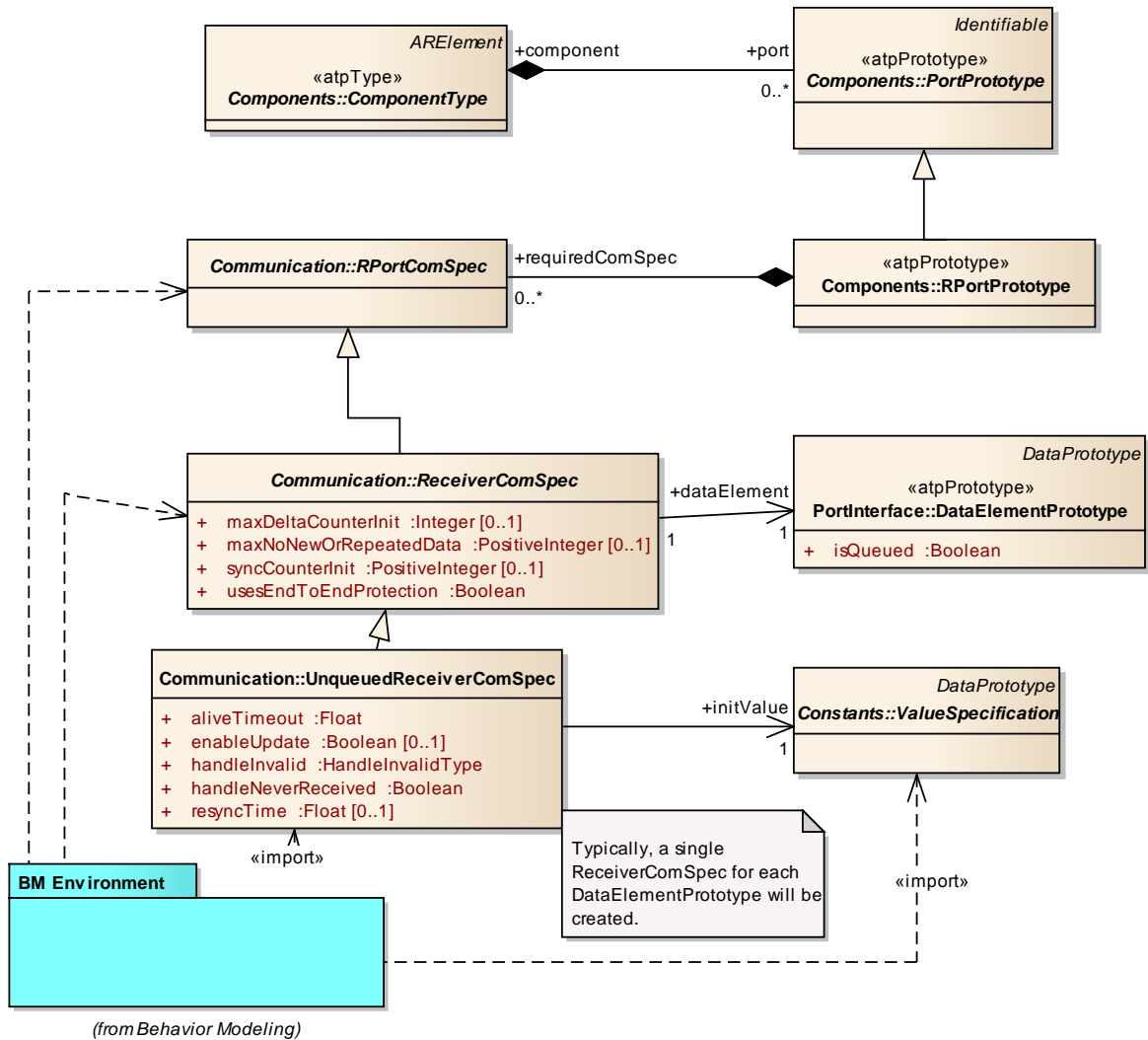


Figure 12-15: Release R3.2.x: ReceiverComSpec (hardcopy from BM ComSpec ReceiverComSpec)

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 7.2.1):

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 _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.	Production	E2EPW_STATUS_OK	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	E2EPW_STATUS_NONEW DATA	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	E2EPW_STATUS_WRONG CRC	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	E2EPW_STATUS_SYNC	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	E2EPW_STATUS_INITIAL	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 _INITIAL, _OK, or _OKSOMELOST.	Production	E2EPW_STATUS_REPEATED	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 _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.	Production	E2EPW_STATUS_OKSOMELOST	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 _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.	Production	E2EPW_STATUS_WRONG SEQUENCE	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 7.2.1):

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	E2EPW_E_DESERIALIZATION	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	E2EPW_E_REDUNDANCY	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”.]()

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.]()

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 extensions on the other side, the set of return values are divided (for each byte) into AUTOSAR use and proprietary use.

[UC_E2E_0304] 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.

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.]()

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:	uint32	E2EPW_Write_<p>_<o>(Rte_Instance <instance>, -- <data>)
Service ID[hex]:	0x00	
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: 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> <p>The byte 1 is the status of runtime checks 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) E2E_E_OK - Function E2EPW_Write completed successfully</p> <p>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</p> <p>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 and then it calls the corresponding RTE_Write function.
---------------------	---

]()

[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>()]()

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> (<instance>
		Rte_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_ReturnType	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.		

]()

[UC_E2E_00301] [The function E2EPW_WriteInit_<p>_<o> shall initialize the E2E_PXXSenderStateType_<p>_<o> with the following values:

Counter = 0]()

12.1.7.1.3 E2EPW_Read_<p>_<o>

[UC_E2E_00165] [

Service name:	E2EPW_Read_<p>_<o>		
Syntax:	uint32	E2EPW_Read_<p>_<o> (<instance> ,
		Rte_Instance	<data>
		--)

Service ID[hex]:	0x00	
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: 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> <p>The byte 1 is the status of runtime checks done within E2E Protection Wrapper function, plus including bit extension checks: 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> <p>The byte 2 is the return value of E2E_PXXCheck function: 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> <p>The byte 3 is the value of E2E_PXXReceiverStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. 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. E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC is incorrect. E2EPW_STATUS_INITIAL - Error: The new data has been received according to communication medium, the CRC is correct, but this is the</p>

	<p>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.

]()

[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.]()

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_ReturnType	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.	

}]()

[UC_E2E_00297] [The function E2EPW_ReadInit_<p>_<o> shall initialize the E2E_PXXReceiverStateType_<p>_<o> with the following values:

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

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:	uint32	E2EPW_Write1_<p>_<o>(Rte_Instance <instance>, -- <data>)
Service ID[hex]:	0x00	
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: 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> <p>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</p> <p>The byte 3 is a placeholder for future use and takes the following values: 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.	

]()

[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.]()

12.1.7.2.2 E2EPW_Write2_<p>_<o>

[UC_E2E_00263][

Service name:	E2EPW_Write2_<p>_<o>	
Syntax:	uint32	E2EPW_Write2_<p>_<o>(Rte_Instance <instance>, -- <data>)
Service ID[hex]:	0x00	
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: 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> <p>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</p> <p>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</p> <p>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.

]()

[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>() .]()

12.1.7.2.3 E2EPW_WriteInit1_<p>_<o>

[SWS_E2E_00318]

Service name:	E2EPW_WriteInit1_<p>_<o>		
Syntax:	Std_ReturnType 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:	Std_ReturnType	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 data structure of redundant channel 1 state data after a detected error or at startup. Note: Redundant channel 1 and channel 2 have a separate instance of E2E_PXXSenderStateType and PXXConfigType.		

⌋()

[SWS_E2E_00322] ⌈ The function E2EPW_WriteInit1_<p>_<o> shall initialize the E2E_PXXSenderStateType_<p>_<o> related to redundant channel 1 with the following values:

Counter = 0. ⌋()

12.1.7.2.4 E2EPW_WriteInit2_<p>_<o>

[SWS_E2E_00319]

Service name:	E2EPW_WriteInit2_<p>_<o>		
Syntax:	Std_ReturnType 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:	Std_ReturnType	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 of redundant channel 2 after a detected error or at startup. Note: Redundant channel 1 and channel 2 have a separate instance of E2E_PXXSenderStateType and PXXConfigType.	

⌋()

[SWS_E2E_00323] ⌈ The function E2EPW_WriteInit2_<p>_<o> shall initialize the E2E_PXXSenderStateType_<p>_<o> related to redundant channel 2 with the following values:

Counter = 0. ⌋()

12.1.7.2.5 E2EPW_Read1_<p>_<o>

[UC_E2E_00265] ⌈

Service name:	E2EPW_Read1_<p>_<o>	
Syntax:	uint32	E2EPW_Read1_<p>_<o>(Rte_Instance <instance>, -- <data>)
Service ID[hex]:	0x00	
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 The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: 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

		<p>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> <p>The byte 2 is the return value of E2E_PXXCheck function: 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> <p>The byte 3 is the value of E2E_PXXReceiverStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. 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. E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC is incorrect. 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. 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. 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. 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. 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 E2EPW_STATUS_SYNC - 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</p>
--	--	---

	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.

]()

[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.]()

12.1.7.2.6 E2EPW_Read2_<p>_<o>

[UC_E2E_00267][

Service name:	E2EPW_Read2_<p>_<o>	
Syntax:	uint32	E2EPW_Read2_<p>_<o>(Rte_Instance <instance>, -- <data>)
Service ID[hex]:	0x00	
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) equal to RTE_E_OK (because Rte_Read is not invoked) The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: 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: 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> <p>The byte 3 is the value of E2E_PXXReceiverStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. 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. E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC is incorrect. 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. 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. 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. 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. 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 E2EPW_STATUS_SYNC - 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.</p>
Description:		The function re-checks the data received with corresponding function Read1 by means of execution of E2E_PXXCheck.

]()

[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.]()

12.1.7.2.7 E2EPW_ReadInit1_<p>_<o>

[SWS_E2E_00320]

Service name:	E2EPW_ReadInit1_<p>_<o>	
Syntax:	Std_ReturnType 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:	Std_ReturnType	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 of redundant channel 1 after a detected error or at startup. Note: Redundant channel 1 and channel 2 have a separate instance of E2E_PXXSenderStateType and PXXConfigType.	

]()

[SWS_E2E_00324] The function E2EPW_ReadInit1_<p>_<o> shall initialize the E2E_PXXReceiverStateType_<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.]()

12.1.7.2.8 E2EPW_ReadInit2_<p>_<o>

[SWS_E2E_00321]

Service name:	E2EPW_ReadInit2_<p>_<o>	
Syntax:	Std_ReturnType 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:	Std_ReturnType	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 of redundant channel 2 after a detected error or at startup. Note: Redundant channel 1 and channel 2 have a separate instance of E2E_PXXSenderStateType and PXXConfigType.	

」()

[SWS_E2E_00325]「 The function E2EPW_ReadInit2_<p>_<o> shall initialize the E2E_PXXReceiverStateType_<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.」()

12.1.8 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`)

However, the variables in the code examples are defined inline and without proper memory mapping, suffix nor compiler abstraction (because when the code examples were introduced, the init function did not yet exist).

To avoid making the code example too complex, the original names for variables are kept (without suffixes). This will be fixed in an upcoming AUTOSAR release.

Note also that the code example does not cover functionality related to the status `E2EPW_STATUS_SYNC`.

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.

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 generated 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)).

12.1.8.1 Code Example – Sender SW-C

12.1.8.1.1 Sender – E2EPW_Write and E2EPW_Write1

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

12.1.8.1.1.1 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;
...
DataType* AppDataEl;

```

Case B (array):

The RTE Generator generates an opaque uint8 array.

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

12.1.8.1.1.2 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;

```

12.1.8.1.1.3 Step S1

Step S1: Application calls E2E Protection Wrapper.

```

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

```

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

```

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

```

12.1.8.1.1.4 Step S2

Step S2: E2E Protection Wrapper (E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()) initializes the data structures used by E2E Library (at first run only). Alternatively, StateVal can be set by an init-function.

```

/* single channel - Write */
static const E2E_P01ConfigType ConfigVal =
    { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
static const E2E_P01ConfigType* Config = &ConfigVal;

static E2E_P01SenderStateType StateVal = {0};
static E2E_P01SenderStateType* State = &StateVal;

```

```
Std_ReturnType E2EPW_WriteInit_<p>_<o>(Rte_Instance Instance) {
    Config    = &ConfigVal;
    StateVal = {0};
    State     = & StateVal;
    Return E2E_E_OK;
}
```

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

```
/* redundant - Write1 */
static const E2E_P01ConfigType ConfigVal1 =
    { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
static const E2E_P01ConfigType* Config1 = &ConfigVal1;

static E2E_P01SenderStateType StateVal1 = {0};
static E2E_P01SenderStateType* State1 = & StateVal1;

Std_ReturnType E2EPW_WriteInit1_<p>_<o>(Rte_Instance Instance) {
    Config1    = &ConfigVal1;
    StateVal1 = {0};
    State1     = & StateVal1;
    Return E2E_E_OK;
}
```

12.1.8.1.1.5 Step S3

Step S3: 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 ISignalIPdu; 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 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 (AppDataEl == NULL) {
    return (E2E_E_INPUTERR_NULL);
}

/* 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. */
static uint8 Data[8] = {0xFF, 0xFF, 0xFF, 0xFF,
                       0xFF, 0xFF, 0xFF, 0xFF};

Data[0] = 0;

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

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

/* low nibble of high byte goes to Data[3] */
Data[3] = (AppDataEl->speed & 0x0F00) >> 8;

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

/* Data[4] is unused but transmitted, so it is explicitly set
to 0xFF*/
Data[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;

```

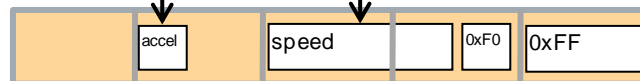


Figure 12-17: Mapping of Data elements into I-PDU

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);
}

```



```
static uint8 Data[8];
memcpy(Data, AppDataE1, 8);
```

12.1.8.1.1.6 Step S4

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

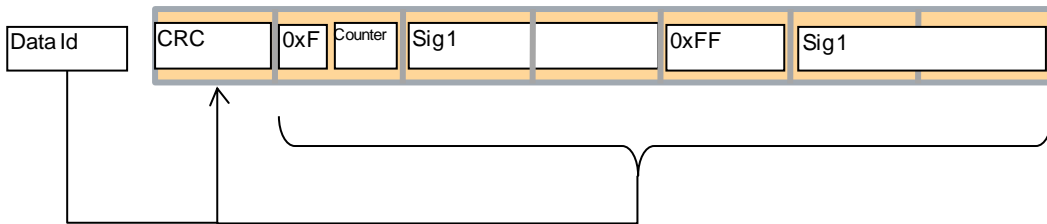
```
/* single channel - Write */
Std_ReturnType retE2EProtect = E2E_P01Protect(Config, State, Data);
```

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

```
/* redundant - Write1 */
Std_ReturnType retE2EProtect = E2E_P01Protect(Config1, State1, Data);
```

12.1.8.1.1.7 Step S5

Step S5: E2E executes protection, updates State and AppDataE1.



CRC := CRC8 over (1) Data Id, (2) all serialized signal (including empty areas, excluding CRC byte itself)

Figure 12-18: Step 4

12.1.8.1.1.8 Step S6

Step S6: The E2E Protection Wrapper (E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()) copies back the control fields to AppDataE1.

Case A (complex data type):

```
AppDataE1->CRC = Data[0]; /* Copy CRC from byte 0 */
AppDataE1->Counter = Data[1]&0x0F; /* Copy counter from byte 1 */
```

This is illustrated by the Figure 12-19:

```
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*/
} AppDataEl;
```

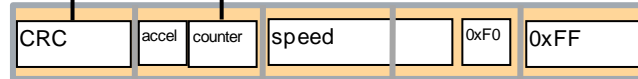


Figure 12-19: Copy back of CRC and alive from I-PDU copy to data element

Case B (array):

```
AppDataEl[0] = Data[0]; /* Copy CRC from byte 0 */
AppDataEl[1] = (AppDataEl[1]&0xF0) | (Data[1]&0x0F); /* Copy CRC */
```

12.1.8.1.1.9 Step S7

Step S7: 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_Write` `<p>` `<o>`) in step S7 does *not* call `Rte_Write` `<p>` `<o>`(`)` function.

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

12.1.8.1.1.10 Step S8

Step S8: The E2E Wrapper creates the return value and returns.

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

12.1.8.1.1.11 Step S9

Step S9: 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(ret);
```

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

12.1.8.1.2 Sender - E2EPW_Write2

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

12.1.8.1.2.1 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)

12.1.8.1.2.2 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);
```

12.1.8.1.2.3 Steps S12

Step S12: E2E Protection Wrapper (`E2EPW_Write2_<p>_<o>()`) initializes the data structures used by E2E Library (at first run only). Alternatively, `StateVal2` can be set by an init-function

```
static const E2E_P01ConfigType ConfigVal2 =
    { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
static const E2E_P01ConfigType* Config2 = &ConfigVal2;

static E2E_P01SenderStateType StateVal2 = {0};
static E2E_P01SenderStateType* State2 = &StateVal2;

Std_ReturnType E2EPW_WriteInit2_<p>_<o>(Rte_Instance Instance) {
    Config2 = &ConfigVal2;
    StateVal2 = {0};
    State2 = &StateVal2;
    Return E2E_E_OK;
}
```

12.1.8.1.2.4 Step S13

The step S13 (of function `E2EPW_Write2_<p>_<o>()`) is 100% identical to Step S3 (of function `E2EPW_Write1_<p>_<o>()`).

12.1.8.1.2.5 Step S14

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

```
/* redundant - Write1 */
Std_ReturnType retE2EProtect = E2E_P01Protect(Config2, State2,
Data);
```

12.1.8.1.2.6 Step S15

The step S15 (of function `E2EPW_Write2_<p>_<o>()`) is 100% identical to Step S5 (of function `E2EPW_Write1_<p>_<o>()`).

12.1.8.1.2.7 Step S16 – skipped

Contrary to step S6, there is no copying back of control fields back to `AppDataE1` in `E2EPW_Write2_<p>_<o>()`.

12.1.8.1.2.8 Steps S17

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):

```
/* error code - voting error between Write1 and Write2.
The error code is different from any code returned by
```

```

E2E_Protect() function */
#define E2EPW_E_REDUNDANCY 0xFF;

if( (AppDataEl->counter != (Data[1] & 0x0F)) ||
    (AppDataEl->crc != (Data[0]          )) )
    plausibilityChecks = E2EPW_E_REDUNDANCY; /* 0xFF */

Std_ReturnType retRteWrite = E2E_E_OK;

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

```

Case B (array):

```

/* error code - voting error between Write1 and Write2 */
#define E2EPW_E_REDUNDANCY 0xFF;

if( ((AppDataEl[1] & 0x0F) != (Data[1] & 0x0F)) ||
    (AppDataEl[0] != (Data[0]          )) )
    plausibilityChecks = E2EPW_E_REDUNDANCY; /* 0xFF */

Std_ReturnType retRteWrite = E2E_E_OK;

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

```

12.1.8.1.2.9 Step S18

Step S18: The E2E Wrapper creates the return value and returns.

```

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

```

12.1.8.1.2.10 Step S19

Step S9: 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(ret2);

```

12.1.8.2 Code Example – Receiver SW-C

12.1.8.2.1 Receiver - E2EPW_Read and E2EPW_Read1

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

12.1.8.2.1.1 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 */
    uint16 speed;      /* 16-bit, but 12 bits used in I-PDU*/
    uint8 accel;       /* 16-bit, but 12 bits used in I-PDU*/
} DataType;
...
DataType* AppDataEl;
```

Case B (array):

The RTE Generator generates an opaque uint8 array.

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

12.1.8.2.1.2 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);
```

12.1.8.2.1.3 Step R2.0

Step R2.0: E2E Protection Wrapper (`E2EPW_Read_<p>_<o>`, `E2EPW_Read1_<p>_<o>()`) initializes the data structures used by E2E Library (at first run only). Alternatively, `stateVal` can be set by an init-function.

```
static const E2E_P01ConfigType ConfigVal =
    { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
static const E2E_P01ConfigType* Config = &ConfigVal;
```

```

static E2E_P01ReceiverStateType StateVal =
    { 0, 0, TRUE, FALSE, 0, E2E_P01STATUS_NONEWDATA };
static E2E_P01ReceiverStateType* State = &StateVal;

Std_ReturnType E2EPW_ReadInit_<p>_<o>(Rte_Instance Instance) {
    Config    = &ConfigVal;
    StateVal = { 0, 0, TRUE, FALSE, 0, E2E_P01STATUS_NONEWDATA };
    State     = &StateVal;
    Return E2E_E_OK;
}

```

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

```

static const E2E_P01ConfigType ConfigVal1 =
    { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
Static const E2E_P01ConfigType* Config1 = &ConfigVal1;

static E2E_P01ReceiverStateType StateVal1 =
    { 0, 0, TRUE, FALSE, 0, E2E_P01STATUS_NONEWDATA };
static E2E_P01ReceiverStateType* State1 = &StateVal1;

Std_ReturnType E2EPW_ReadInit1_<p>_<o>(Rte_Instance Instance) {
    Config1    = &ConfigVal1;
    StateVal1 = { 0, 0, TRUE, FALSE, 0, E2E_P01STATUS_NONEWDATA };
    State1     = &StateVal1;
    Return E2E_E_OK;
}

```

12.1.8.2.1.4 Step R2.1

Step R2.1: Wrapper (E2EPW_Read_<p>_<o>, E2EPW_Read1_<p>_<o>()) checks the parameters and then calls RTE function Rte_Read to receive the data element.

```

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

retRteRead = Rte_Read_<p>_<o>(Instance, AppDataE1);

State->NewDataAvailable = (retRteRead == RTE_E_OK) ? TRUE : FALSE;

```

12.1.8.2.1.5 Step R3

Step R3: 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 */
static uint8 Data[8];

Data[0] = 0;

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

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

/* low nibble of high byte goes to Data[3] */
Data[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[3] |= 0xF0;

/* Data[4] is unused but transmitted, so it is explicitly set
   to 0xFF*/
Data[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:

```

static uint8 Data[8];
/* Copy from AppDataE1 to Data */
memcpy(Data, AppDataE1, 8);

```

12.1.8.2.1.6 Step R4

Step R4: E2E Protection Wrapper calls the E2E library to check the data element.

```

/* single channel - Read */
Std_ReturnType retE2ECheck = E2E_P01Check(Config, State, Data);

```

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

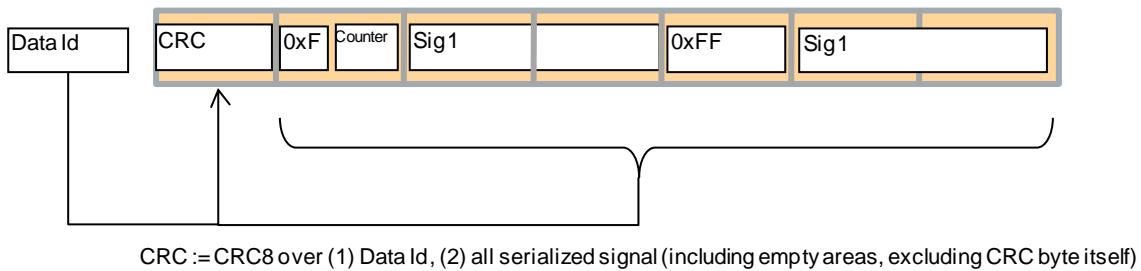
```

/* redundant - Read1 */
Std_ReturnType retE2ECheck = E2E_P01Check(Config1, State1, Data);

```

12.1.8.2.1.7 Step R5

Step R5: E2E computes CRC, and executes the checks.



12.1.8.2.1.8 Step R6 – skipped

No control fields need to be copied to `AppDataE1`, as they are only verified.

12.1.8.2.1.9 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.

```

/* 1 if COM/RTE did not correctly expand I-PDU into data elements */
/* Value 1 is reserved for the Wrapper, E2E Library Check function
does not return it. */
#define E2EPW_E_DESERIALIZATION 1

...
Std_ReturnType plausibilityChecks = E2E_E_OK;

/* 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

```
Std_ReturnType plausibilityChecks = 0;
```

12.1.8.2.1.10 Step R8

Step R8: The E2E wrapper returns to the application.

```
/* single channel */
return ( (retRteRead) | (retE2ECheck<<8) |
         (plausibilityChecks<<16) | (uint32) (State->Status)<<24 );
```

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

```
/* redundant */
return ( (retRteRead) | (retE2ECheck<<8) |
         (plausibilityChecks<<16) | (uint32) (State1->Status)<<24 );
```

12.1.8.2.1.11 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(ret);
}

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(ret);
}

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(ret);
}

targetSpeed = (AppDataEl[2]) | (AppDataEl[3]<<8 & 0x0F);
targetAccel = AppDataEl[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(ret);
}

if(wrapperRet1 != 0) swc_error_handler(ret1);
targetSpeed1 = (AppDataEl[2]) | (AppDataEl[3]<<8 & 0x0F);
targetAccel1 = AppDataEl[1] >> 4;

```

12.1.8.2.2 Receiver - E2EPW_Read2

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

12.1.8.2.2.1 Step R10 – skipped

Value unused to numbering consistency.

12.1.8.2.2.2 Step R11

Step R11: Application calls the wrapper again.

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

12.1.8.2.2.3 Step R12.0

Step R12.0: E2E Protection Wrapper (`E2EPW_Read2_<p>_<o>()`) initializes the data structures used by E2E Library (at first run only). Alternatively, `stateVal` can be set by an init-function.

```

static const E2E_P01ConfigType ConfigVal2 =
    { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
Static const E2E_P01ConfigType* Config2 = &ConfigVal2;

static E2E_P01ReceiverStateType StateVal2 =
    { 0, 0, TRUE, FALSE, 0, E2E_P01STATUS_NONEWDATA };
static E2E_P01ReceiverStateType* State2 = &StateVal2;

```

```

Std_ReturnType E2EPW_ReadInit2_<p>_<o>(Rte_Instance Instance) {
    Config2    = &ConfigVal2;
    StateVal2 = { 0, 0, TRUE, FALSE, 0, E2E_P01STATUS_NONEWDATA };
    State2     = &StateVal2;
    Return E2E_E_OK;
}

```

12.1.8.2.2.4 Step R12.1 – partially skipped

Contrary to Step R2.1, RTE is not read. Both read steps use the same data from RTE, which are read in step Step R2.1. There is only checking for parameters and setting of State->NewDataAvailable:

```

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

/* set always to true, because Rte_Read is not invoked. */
State->NewDataAvailable = TRUE;

```

12.1.8.2.2.5 Steps R13

The step R13 (of function E2EPW_Read2_<p>_<o>()) is 100% identical to Step R3 (of function E2EPW_Read1_<p>_<o>()).

12.1.8.2.2.6 Step R14

Step R14: E2E Protection Wrapper calls the E2E library to check the data element.

```

Std_ReturnType retE2ECheck = E2E_P01Check(Config2, State2, Data);

```

12.1.8.2.2.7 Step R15

The step R14 (of function E2EPW_Read2_<p>_<o>()) is 100% identical to Step R5 (of function E2EPW_Read1_<p>_<o>()).

12.1.8.2.2.8 Step R16 – skipped

No control fields need to be copied to AppDataE1, as they are only verified by the wrapper.

12.1.8.2.2.9 Step R17

The step R17 (of function E2EPW_Read2_<p>_<o>()) are 100% identical to step 7 (of function E2EPW_Read1_<p>_<o>()).

12.1.8.2.2.10 Step R18

Step R8: The E2E wrapper returns to the application.

```
return ( (retRteRead) | (retE2ECheck<<8) |
        (plausibilityChecks<<16) | (uint32)(State2->Status)<<24 );
```

12.1.8.2.2.11 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 */
targetSpeed2 = AppDataE1->speed;
targetAccel2 = AppDataE1->accel;

/* check if E2EPW_Read2 was successful */
if(wrapperRet2 != 0) swc_error_handler(ret2);

/* 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(ret1, ret2);

/* check for corruption of AppDataE1 after CRC has been checked */
if(targetSpeed2 != targetSpeed1) swc_error_handlerR(ret1, ret2);
if(targetAccel2 != targetAccel1) swc_error_handlerR(ret1, ret2);

/* consume targetSpeed1/targetSpeed2 and targetAccel1/targetAccel2*/
```

Case B (array):

```
/* copy values from data element */
targetSpeed2 = (AppDataE1[2]) | (AppDataE1[3]<<8 & 0x0F);
targetAccel2 = AppDataE1[1] >> 4;

/* check if E2EPW_Read2 was successful */
if(wrapperRet2 != 0) swc_error_handler(ret2);

/* 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(ret1, ret2);
```

```
/* check for corruption of AppDataE1 after CRC has been checked */  
if(targetSpeed2 != targetSpeed1) swc_error_handlerR(ret1, ret2);  
if(targetAccel2 != targetAccel1) swc_error_handlerR(ret1, ret2);  
  
/* consume targetSpeed1/targetSpeed2 and targetAccel1/targetAccel2*/
```

12.2 COM 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.

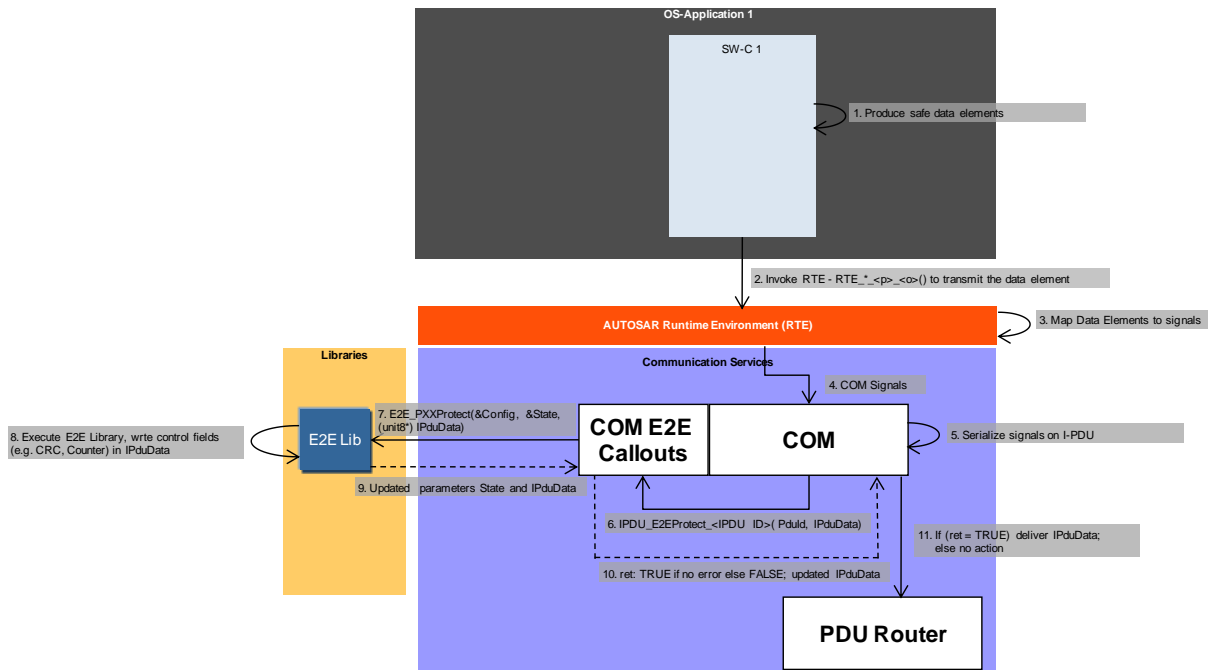


Figure 12-20: Callout – overall flow – P-port

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_PXXReceiverStateType). 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.

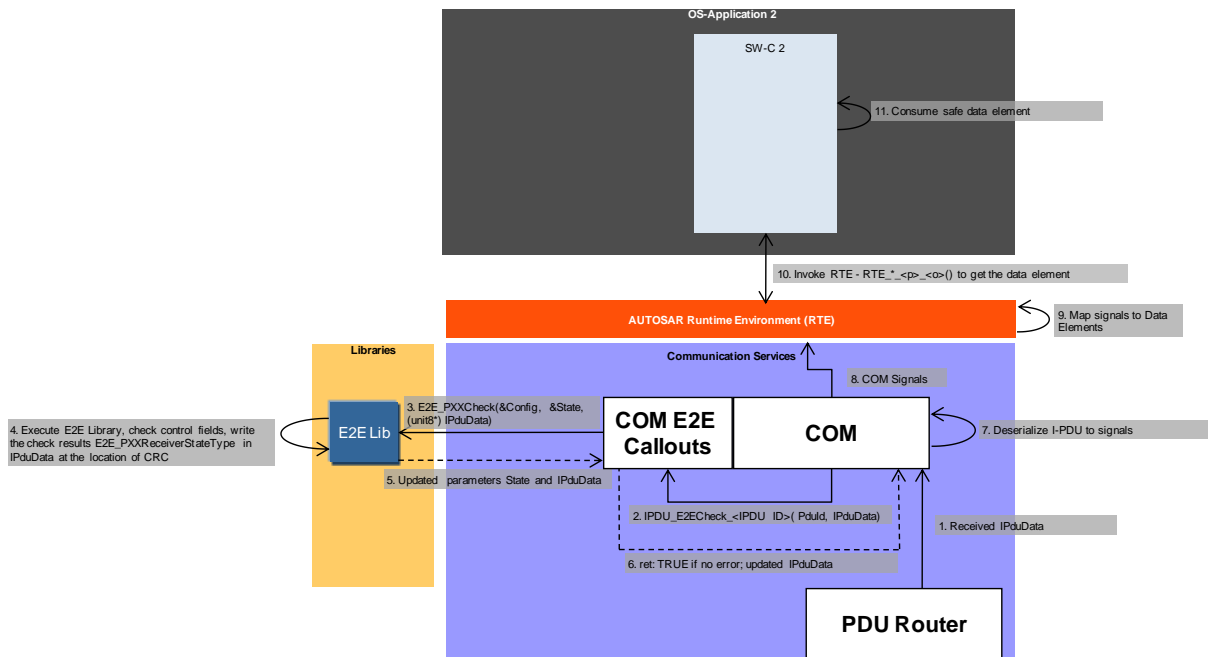


Figure 12-21: Callout – overall flow – R-port

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.

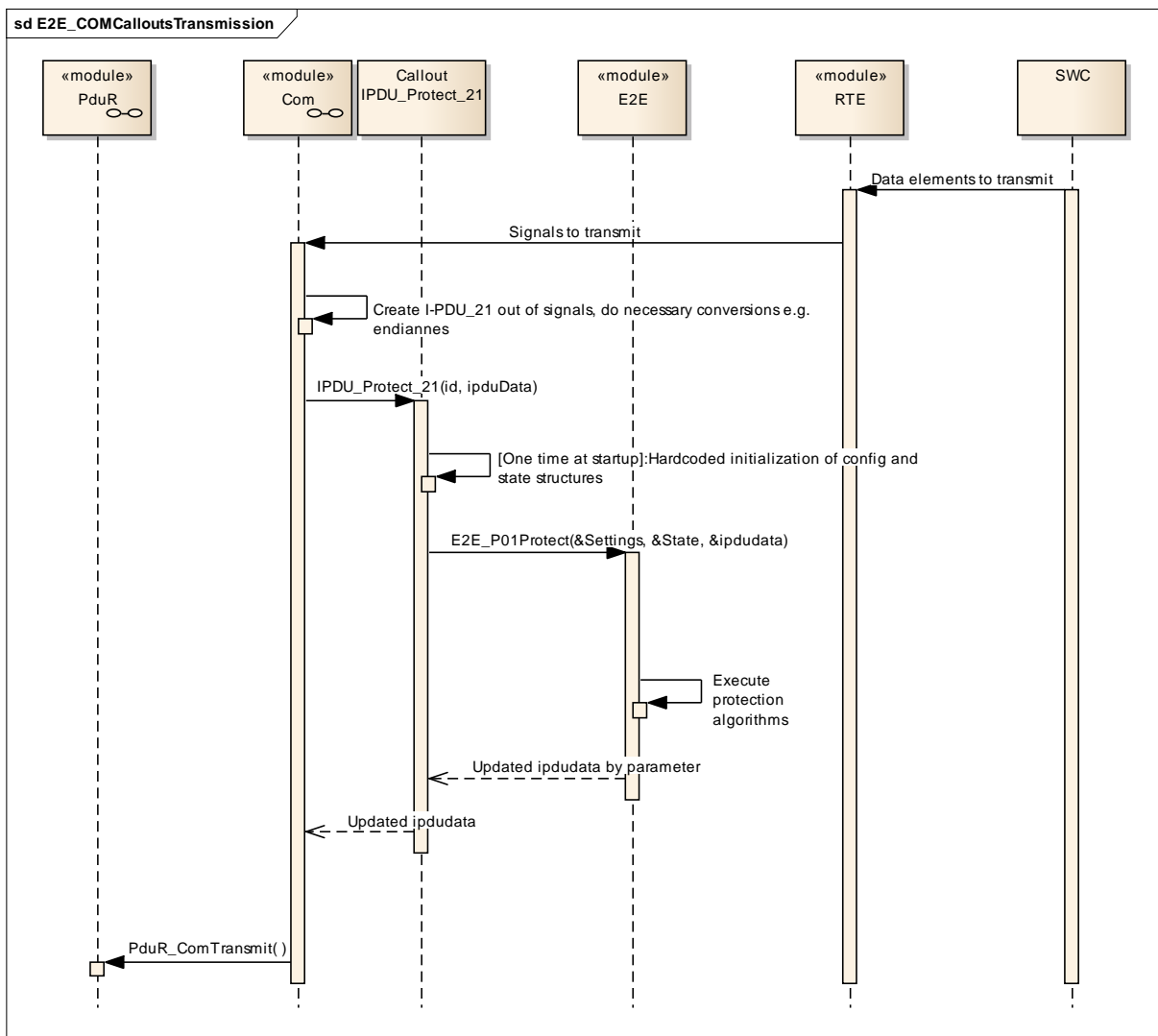


Figure 12-22: Callout – sequence – sending

According to COM SWS, the callouts shall conform to the following syntax:
 FUNC(boolean, COM_APPL_CODE) <IPDU_CalloutName> (PduIdType id, P2VAR (uint8, AUTOMATIC, COM_VAR_NOINIT) ipduData)

[UC_E2E_00250] [The transmission callout for usage with E2E shall be the following:
IPDU_E2EProtect_<IPDU ID>(PduIdType id, P2VAR (uint8, AUTOMATIC,
COM_VAR_NOINIT) ipduData).

For example, the callout to protect the I-PDU with handle 21 shall have the name
IPDU_E2EProtect_21()]()

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.

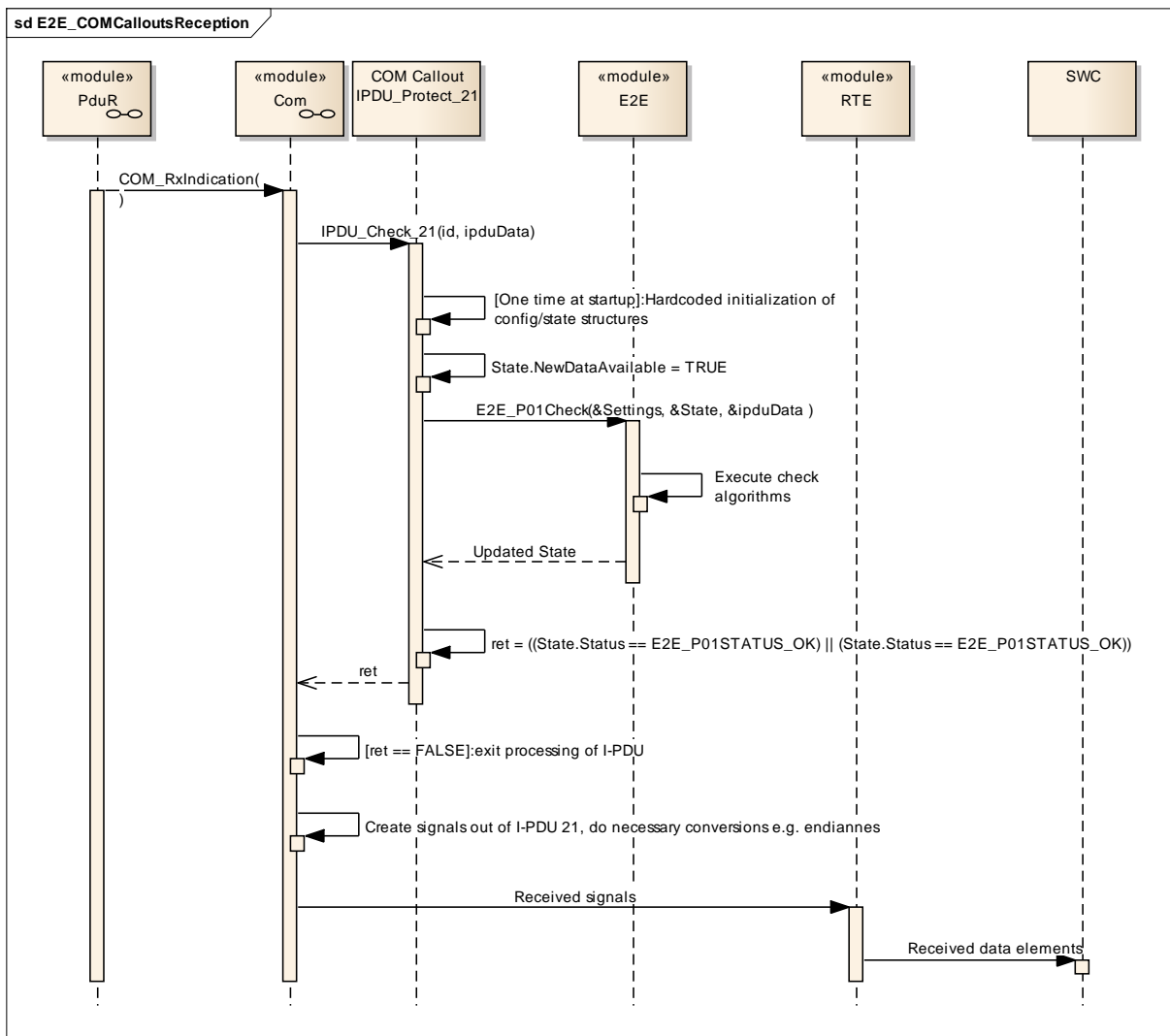


Figure 12-23: Callout - sequence - reception

[UC_E2E_00251][The reception callout for usage with E2E shall be the following:
IPDU_E2Echeck_<IPDU ID> () .

For example, the callout to protect the signal groups in an I-PDU with handle 21 shall have the name IPDU_E2Echeck_21 ()]()

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 [12], Software Component Template [11] and ECU configuration [13]. 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.]()

[UC_E2E_00290][If the E2EProtection is done via COM Callouts then the `EndToEndProtectionISignalIPdu` shall be defined.]()

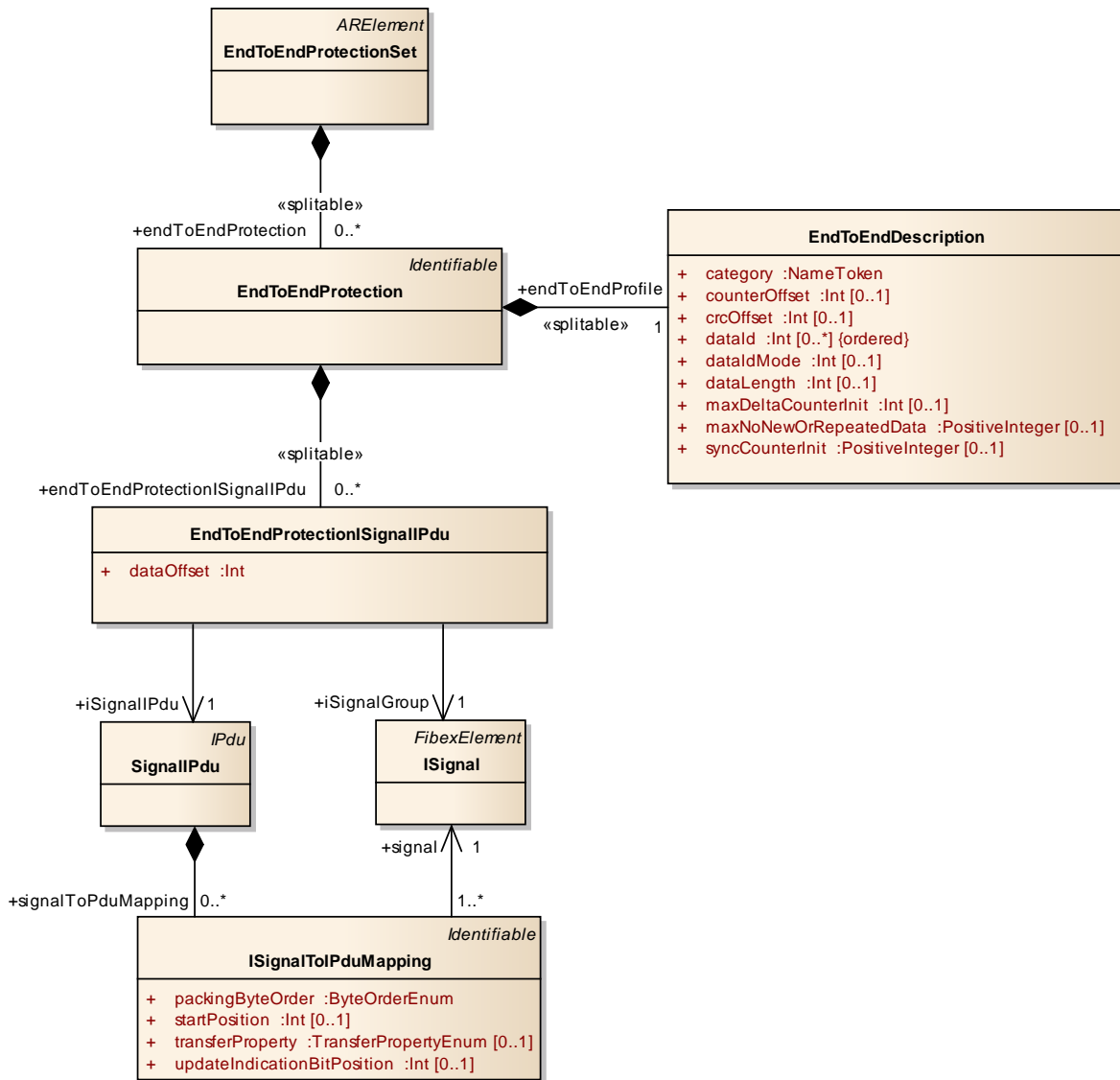


Figure 12-24: Release R3.2.x: COM Callouts Configuration (hardcopy from DOC_PduEndToEndProtection)

Note that in R3.2 (contrary to >=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. EndToEndProtectionISignallPdu: association of one E2E protection to a one I-PDU and to one signal group,
 - a. EndToEndProtectionISignallPdu.dataOffset: offset in bits of the signal group in the I-PDU.

ISignallPdu.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 id, P2VAR
(uint8, AUTOMATIC, COM_VAR_NOINIT) ipduData){

    /* 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_P01SenderStateType Sta_Write_21 = {0};

    Std_ReturnType ret = E2E_P01Protect(& Cfg_Write_,
        & Sta_Write_21,
        & ipduData);

    /* return TRUE if no error in protect function */
    return (ret != 0);

}

```

Receiver

```

FUNC(boolean, COM_APPL_CODE) IPDU_E2ECheck_21 (PduIdType id, P2CONST
(uint8, AUTOMATIC, COM_VAR_NOINIT) ipduData) {

    /* 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_P01ReceiverStateType 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, ipduData);

/* 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.

12.4 Examples for the implementation of E2E protection concepts based on E2E-Library

In the following chapter exemplary principles and approaches for E2E protection concepts based on E2E-Library are provided.

An E2E protection concept is more than only adding adequate safety mechanisms to data elements (e.g. using E2E Profile 1 or 2).

To ensure the integrity of a communication channel with the required safety integrity level the E2E protection concept needs to consider the safety-related properties of the data transmitted from the sender to the receiver(s) that require protection (e.g. correctness, consistency, completeness, timeliness or availability of data).

In order to implement an E2E protection concept that focuses on the protection of correctness, consistency, completeness, timeliness and the detection of non-availability of data, its principles are provided in this chapter.

Note: For an E2E protection concept that focuses on ensuring the availability of data an implementation of the communication channel, with a sufficient fault tolerance is

needed (e.g. using independent redundant channels). The usage of redundant communication channels may create a need for additional safety mechanisms e.g. to ensure the consistency of the data streams when transmitted independently.

12.4.1 Basic principles

Typical basic principles for effective E2E protection concepts are:

- In normal operation mode, the sender ensures that it sends out valid data on a regular basis (e.g. cyclic).
- In this context valid data can be:
 - Data fully complying with their required safety-related properties;
 - Data complying with their required safety-related properties to the extent signaled by an additionally provided qualifier (i.e. signal qualifier);
 - Data explicitly labeled as invalid data (e.g. using an signal invalid value)
- In normal operation mode, the sender groups the data as pre-determined (e.g. to ensure consistency for a set of data) and protects the grouped data with suitable protection mechanisms (e.g. by using the protect functions provided by E2E-Library) prior to their transmission.
- In case of an internal fault, the sender ensures that it sends out either data explicitly labeled as invalid (i.e. only the specific data elements that are possibly affected by this internal fault) or else no data (i.e. fail-safe respective fail-silent behavior of sender in case of a severe fault).
- The infrastructure used for data transmission from a sender to the receiver(s) (e.g. BSWM, Buses, Gateways, etc.) is designed and implemented in such way that it cannot systematically interfere with the used E2E-protection (e.g. by unpacking protected data including the re-calculation of their CRC).
- In normal operation mode, the receiver monitors whether new data has arrived on a regular basis (e.g. cyclic) independently from an external trigger condition coming from elements to which it wants to achieve freedom from interference (e.g. COM).
- In normal operation mode, the receiver is able to detect relevant communication faults within its determined time interval by evaluating the protection mechanisms of the received data and its internal timeout monitoring.
- In case of an detected communication fault, the receiver autonomously realizes the necessary reactions to mitigate the detected communication fault within its determined time interval in compliance with the functional safety concept of the system (i.e. fail-safe respective fail-silent behavior of receiver)
- The fault tolerance time interval of the respective safety-related system is not violated when adding up the allowed time interval for the detection and mitigation of faults at the sender, the time interval required for robustness of data transmission during normal operation (e.g. to compensate gateways) and the allowed time interval for the detection and mitigation of faults at the receiver.

12.4.2 Determination of the integrity of a communication channel within the receiver

To determine the integrity of communication and to distinguish if the received data are valid the receiver (e.g. a SWC) can:

- evaluate each received protected data (e.g. by using the check functions provided by E2E-Library)
- evaluate all protected data it received within its determined time interval for error detection and qualification t_{EDQ} up to the data it received at last.

To evaluate both aspects for the determination of communication integrity a receiver can implement a monitoring function as shown in Figure 12-26:

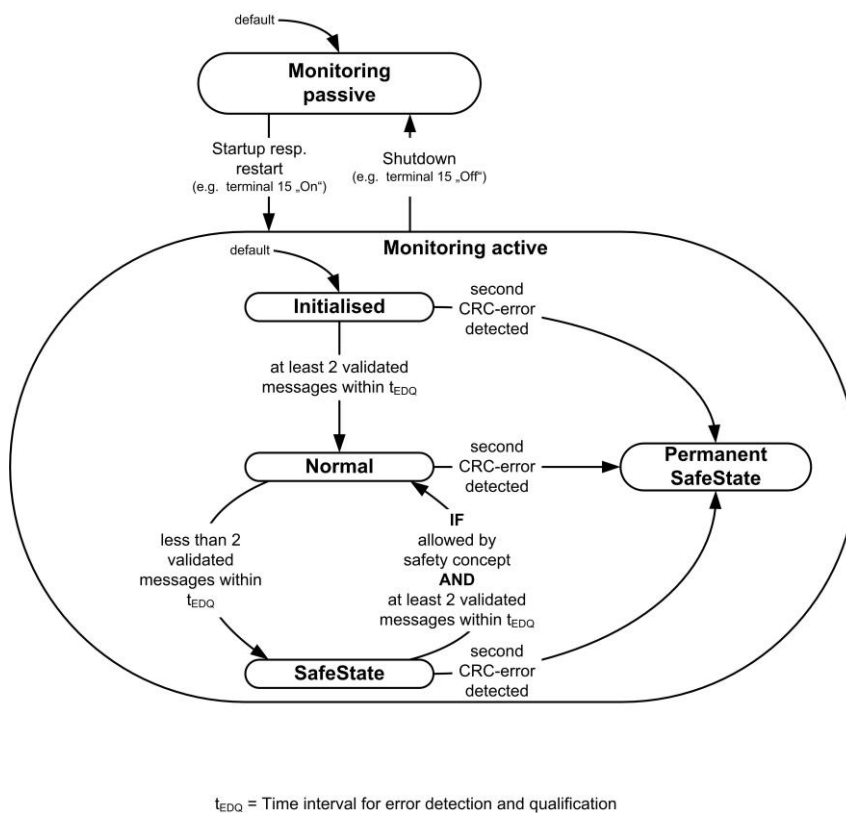


Figure 12-26: Example for a monitoring function to determine the integrity of communication within a receiver

To implement this monitoring function the receiver creates a history of the data it received.

Received valid data (i.e. status of check function is e.g. E2EPW_STATUS_OK or E2EPW_STATUS_OKSOMELOST) is stored with a history as follows:

- Generation 0 is the latest (up to date) received valid data
- Generation 1 is the second-latest received valid data
- Generation 2 is the third-latest received valid data
- etc.

To do so, each recently received valid message is stored as Generation 0 having a reference value indicating its age set to 0.

Every time the receiver checks for the arrival of new data it increments the age of its already received data by 1. Stored data can be used as basis for a safety-related functionality provided by the receiver as long as its age reference value is less a determined boundary value N . The parameter N can be derived by dividing the determined time interval for error detection and qualification t_{EDQ} with the cycle time used for its regular transmission (e.g. for a receiver having a $t_{EDQ} = 160\text{ms}$ and a regular cycle time of 20ms the value $N = 160\text{ms}/20\text{ms} = 8$).

In case that sufficiently up to date data is no longer available, the receiver carries out the reaction determined in the safety concept. Such reaction can be a temporary or a permanent safe state. Depending on the systems functional needs or its safety-related properties to be protected a different condition to enable switching from Initialised to Normal or SafeState instead of „less than 2“ may be adequate.

In contrast to errors indicated based on the evaluation of the counter - CRC-errors are unlikely to be a „false alarm“ (e.g. when using a good CRC-polynomial a detected CRC-error indicates that a data corruption occurred).

Considering this fact, it is implausible that a stream of data transmitted from a sender to a receiver without any detected CRC-error contains a significant number of undetected corrupted data.

Due to this a more stringent reaction upon CRC-errors is adequate, because from the detection of the first CRC-error on the subsequent data stream may contain a significant number of undetected corrupted data if it continues to also contain a significant number of CRC-errors.

Without any limitation of the maximum number of CRC-errors a receiver will tolerate before reacting upon such a questionable overall integrity of its used communication channel (e.g. transition into a permanent safe state if the second CRC-error is detected), the probability that more than one undetected erroneous data will be received within its time interval for error detection and qualification (t_{EDQ}) cannot be neglected in general any more.

The fault tolerance designed into the receiver (see UC_E2E_00170) may be exceeded as a possible consequence.

13 Usage and generation of DataIDLists for E2E profile 2

An appropriate selection of DataIDs for the DataIDList in E2E Profile 2 allows increasing the number of messages for which detection of masquerading is possible. The DataID is used when calculating the CRC checksum of a message, whereas the DataID is not part of the transmitted message itself, i.e. the message received by the receiver does not contain this information.

Any receiver of the intended message needs to know the DataID a priori. The performed check of the received CRC at the receiver side does only match if and only if the assumed DataID on the receiver side is identical to the DataID used at the sender side.

Thus, the DataID allows protecting messages against masquerading. It is important that the used DataID is known solely by the intended sender and the intended receiver.

With a constant DataID (independent of the Counter) the maximum number of messages that can be protected independently using E2E Profile 2 is limited by the length of the CRC (i.e. with a CRC length of 8 bits the number of independent DataID is $2^8 = 256$, this equates to the maximum number of independent messages for detection of masquerading).

However, E2E Profile 2 uses a method to allow more messages to be protected against masquerading by exploiting the prerequisite that a single erroneously received message content does not violate the safety goal (a basic assumption taken in the design of applications of receiving SW-Cs).

The basic idea in E2E Profile 2 is to use a DataIDList with several DataIDs that are selected in a dynamic behavior for the calculation of the CRC checksum. The DataID is determined by selecting one element out of DataIDList, using the value of Counter as an index (for detailed description see E2E profile 2).

The examples given below were selected to show two exemplary use cases. It is demonstrated how the detection of masquerading is performed.

Although the examples take some assumptions on the configuration, the argumentation is valid without loss of generality. For sake of simplicity, these additional constraints are not explained in the following examples.

13.1 Example A (persistent routing error)

Assumptions

Consider a network with one or more nodes as sender (messages A to F) and one node as the intended receiver of the safety relevant message (message B). The messages are configured to use the DataIDList as shown in Figure 13-1 and Figure 13-2.

Sender-ECU		DataIDList															
		DataID for Counter =															
message		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sender A		177	103	29	206	132	58	235	161	87	13	190	116	42	219	145	71
Sender B		146	41	187	82	228	123	18	164	59	205	100	246	141	36	182	77
Sender C		102	204	55	157	8	110	212	63	165	16	118	220	71	173	24	126
Sender D		225	199	173	147	121	95	69	43	17	242	216	190	164	138	112	86
Sender E		181	112	43	225	156	87	18	200	131	62	244	175	106	37	219	150
Sender F		244	244	244	244	244	244	244	244	244	244	244	244	244	244	244	244

←special case of static DataID

Figure 13-1: Sender ECU IDs

Receiver-ECU		DataIDList															
		Counter =															
message		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Receiver B		146	41	187	82	228	123	18	164	59	205	100	246	141	36	182	77

Figure 13-2: Sender ECU IDs

In the example of Figure 13-3 it is assumed that a routing error occurs at a specific point in time. All messages are of same length. The routing error persists until it is detected. For instance a bit flip of the routing table in a gateway could lead to such a constant misrouting. It is further assumed that the senders of messages B and E have the same sequence counter (worst case situation for detection in the receiver).

The receiver should only receive message B and expects therefore the DataIDs of DataIDList of message B. Every time the expected DataID matches with the used DataID in the CRC-protected message, the result of the CRC check will be *valid*. In any other case the CRC checksum in the message differs from the expected CRC result and the outcome of the CRC check is *not valid*.

Solution

As depicted, the first routing error occurs when both senders reach Counter = 6. Since the DataIDList in both senders have DataID = 18 for Counter = 6, the receiver will not detect the erroneously routed message of sender E. However, for any other Counter the values of DataIDs do not match, thus the CRC check in the receiver will be *not valid*.

With this, it is obvious that the misrouting is detected at least for the second received misrouted message (even if some messages were not received at all).

Sender of B		Sender of E		Receiver expects message B				
Counter	DataID	Counter	DataID	Counter	DataID used	check	DataID expected	result of CRC-Check
0	146	0	181	0	146	=	146	valid
1	41	1	112	1	41	=	41	valid
2	187	2	43	2	187	=	187	valid
3	82	3	225	3	82	=	82	valid
4	228	4	156	4	228	=	228	valid
5	123	5	87	5	123	=	123	valid
here 1 st →	6	6	18	6	18	=	18	erroneously undetected! (valid)
routing error	7	7	200	7	200	≠	164	error detected (not valid)
	8	8	131	8	131	≠	59	error detected (not valid)
	9	9	62	9	62	≠	205	error detected (not valid)
	10	10	244	10	244	≠	100	error detected (not valid)
	11	11	175	11	175	≠	246	error detected (not valid)
	12	12	106	12	106	≠	141	error detected (not valid)
	13	13	37	13	37	≠	36	error detected (not valid)
	14	14	219	14	219	≠	182	error detected (not valid)
	15	15	150	15	150	≠	77	error detected (not valid)

	5	5	87	5	87	≠	123	error detected (not valid)

Figure 13-3: example A configuration

13.2 Example B (forbidden configuration)

Not every DataIDList is allowed to be used for every message length. A short explanation to demonstrate this is shown in this example.

Consider a message G with a total length of 8 bytes. Both, sender and receiver are configured to use the DataIDList depicted in Figure 13-4.

Receiver-ECU		DataIDList															
		Counter =															
message		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Receiver	G	73	144	215	35	106	177	248	68	139	210	30	101	172	243	63	134

Figure 13-4: forbidden configuration

Without loss of generality the payload is assumed to be [22,33,44,55,66,77].

For the defined CRC generator polynomial in profile 2 the CRC checksums are as follows:

Counter	DataID	CRC-result
CRC(0, 22, 33, 44, 55, 66, 77, 73)		= 114
CRC(1, 22, 33, 44, 55, 66, 77, 144)		= 197
CRC(2, 22, 33, 44, 55, 66, 77, 215)		= 66
CRC(3, 22, 33, 44, 55, 66, 77, 35)		= 66
CRC(4, 22, 33, 44, 55, 66, 77, 106)		= 207
CRC(5, 22, 33, 44, 55, 66, 77, 177)		= 38
CRC(6, 22, 33, 44, 55, 66, 77, 248)		= 20
CRC(7, 22, 33, 44, 55, 66, 77, 68)		= 165
CRC(8, 22, 33, 44, 55, 66, 77, 139)		= 120
CRC(9, 22, 33, 44, 55, 66, 77, 210)		= 44
CRC(10, 22, 33, 44, 55, 66, 77, 30)		= 110
CRC(11, 22, 33, 44, 55, 66, 77, 101)		= 23
CRC(12, 22, 33, 44, 55, 66, 77, 172)		= 121
CRC(13, 22, 33, 44, 55, 66, 77, 243)		= 207
CRC(14, 22, 33, 44, 55, 66, 77, 63)		= 141
CRC(15, 22, 33, 44, 55, 66, 77, 134)		= 175

One can see that DataID = 215 for Counter = 2 leads to the same CRC checksum as DataID = 35 for Counter = 3. Moreover, DataID = 106 for Counter = 4 leads to the same CRC checksum as DataID = 243 for Counter = 13.

A routing error of a non-CRC-protected message with constant payload and a sequence counter could be undetected at the receiver side if

1. the first routing error occurs at Counter = 2 and is persistent, or
2. the routing error occurs only at Counter = 4 and Counter = 13.

In both cases the second masquerading error is not detected.

Thus, the considered DataIDList of message G in Figure 13-4 *must not* be used for messages with a total length of 8 bytes. (Remember: the DataID itself is never transmitted on the bus).

13.3 Conclusion

The proposed method with dynamic DataIDs for CRC calculation allows protecting significantly (several orders of magnitude) more messages against masquerading than with a static DataID.

The set of DataIDList needs to be generated with appropriate care to utilize the strength of the shown method. Every DataIDList is only allowed to be assigned once to a message within the network/system. The message length needs to be considered in the assignment process since not every DataIDList is allowed to be used for every message length.

13.4 DataIDList example

This section presents an part of exemplary DataIDList. The example has 500 lines, which means that this enables to identify 500 different data.

This DataIDList has been selected and tested with appropriate care to comply with current safety standards. Every user of the provided DataIDLists is responsible to check if the following list is suitable to fulfill his constraints of the intended target network.

14 Not applicable requirements

[SWS_E2E_00294][These requirements are not applicable to this specification.](SRS_BSW_00338, SRS_BSW_00168, SRS_BSW_00375, SRS_BSW_00339, SRS_BSW_00369, SRS_BSW_00336, SRS_BSW_00435)