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 Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Change Description
4.2.2	AUTOSAR Release Management	• Introduced new E2E state machine profile status E2E_P_NONEWDATA. Adapted figures, API tables and mapping functions. This solves an issue with deterministic startup of the state machine. • Updated Figure 7-7, added behavior in case ReceivedCounter is out of range. • Assigned new specification ID SWS_E2E_00478 to duplicate specification SWS_E2E_00324 (specification of profile 4). • Fixed figure 7-6 “Calculate CRC over Data ID and Data”, which was already fixed in R4.1.2 but falsely included as of R4.1.1.
4.2.1	AUTOSAR Release Management	• Introduction of E2E profiles 4, 5, 6 • Introduction of E2E state machine • Introduction of init functions and status mapping functions for profiles 1, 2 • Overview of wrapper, by means of several new diagrams.
4.1.3	AUTOSAR Release Management	• Editorial changes
4.1.2	AUTOSAR Release Management	• Correction in E2E variant 1C • Various minor corrections • Editorial changes
4.1.1	AUTOSAR Release Management	• Full support for E2E protection at signal group level • Removed dependency to Rte_IsUpdated • Changed recommendations about the maximum data lengths

Document Change History		
Release	**Changed by**	**Change Description**
		• Addition of initialization functions to the redundant wrapper • Corrections in code examples
4.1.1	AUTOSAR Administration	• 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
4.0.3	AUTOSAR Administration	• 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
3.1.5	AUTOSAR Administration	• Corrected the wrapper configuration. • Corrected the code example for the usage of the wrapper.
3.1.4	AUTOSAR Administration	• Initial Release

Table of Contents

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).

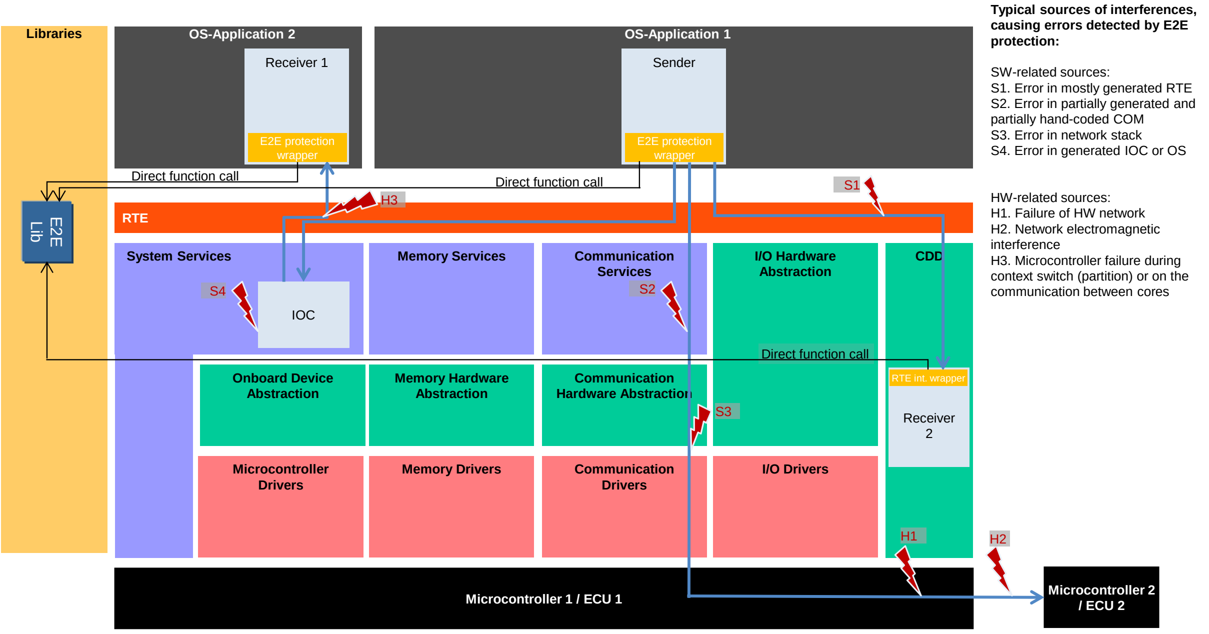


Figure 1-1: Example of faults mitigated by E2E protection

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).

The E2E library is invoked from:

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

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

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

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

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

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

2 Acronyms and abbreviations

All technical terms used in this document, except the ones listed in the table below, can be found in the official AUTOSAR glossary [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. 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, 02, 04, 05, 06 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 2 has in R4.2.1 a new setting offset. This offset can be conigured in the system template. However, the E2E Profile 2 specification does not support the case when offset is different than 0. The specification of E2E Profile 2 will be fixed in a future AUTOSAR release, to support a configurable offset.

E2E Profile 1 in the “Double Data ID configuration” uses an implicit 2-byte Data ID, over which CRC8 is calculated. As a CRC over two different 2-byte numbers may result with the same CRC, some precautions must be taken by the user. See UC_E2E_00072 and UC_E2E_00073.

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

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. ⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08528)

[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 be protected by E2E-Library are mapped for data transmission.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

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, were information is received more than once.

4.3.3.2 Loss of information

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

4.3.3.4 Insertion of information

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

4.3.3.5 Masquerading

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

4.3.3.6 Incorrect addressing

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

4.3.3.11 Blocking access to a communication channel

A type of communication fault, were 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_E2E_08527)

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

[SWS_E2E_00311]⌈ The configuration of the E2E Library and of the code invoking it (e.g. E2E wrapper 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.⌋ (SRS_E2E_08528)

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.

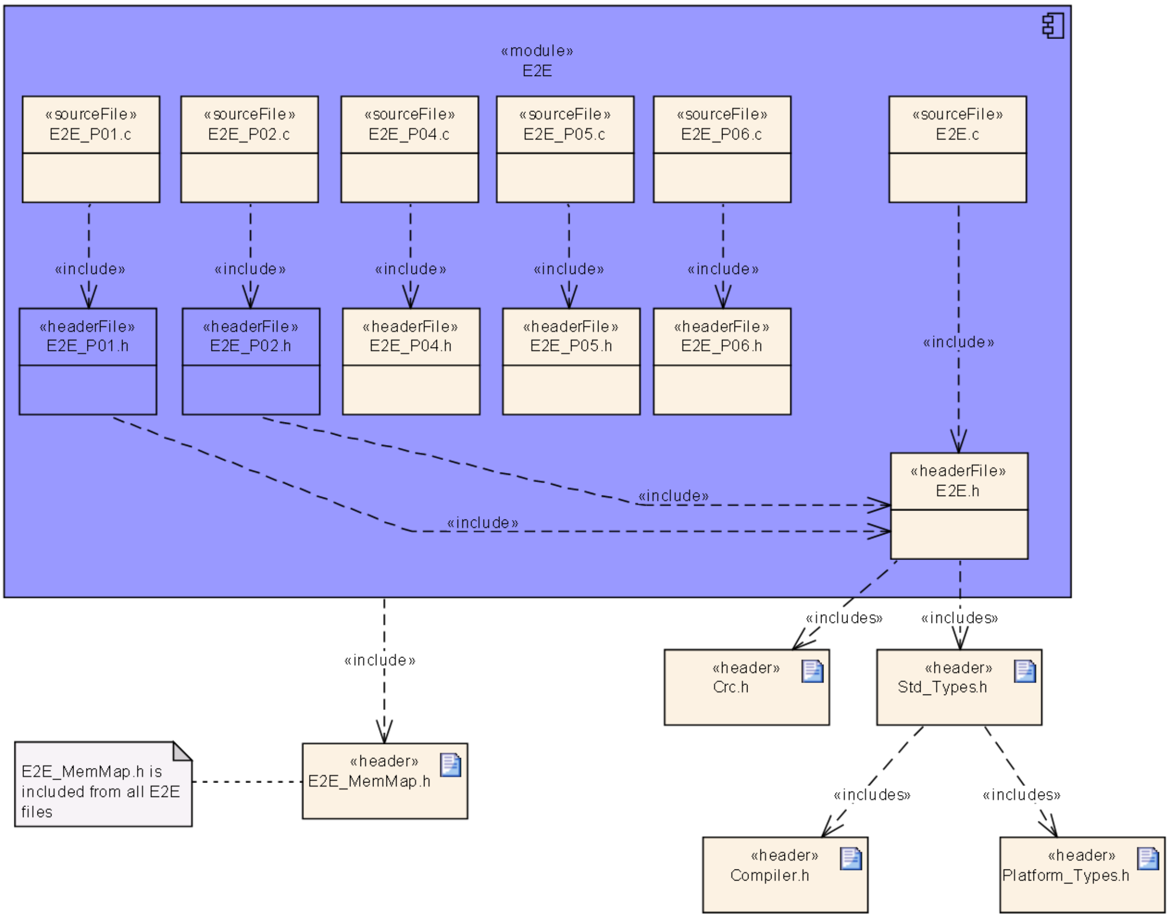


Figure 5-1: File dependencies

[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: e.g. 01, 02, …representing the profile) and E2E_SM.c and E2E_SM.h (for E2E state machine).⌋ (SRS_E2E_08528)

[SWS_E2E_00215]⌈ Files E2E_PXX.c and E2E_PXX.h shall contain implementation parts specific of each profile.⌋ (SRS_E2E_08528)

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

[SWS_E2E_00115]⌈ E2E library files (i.e. E2E_*.*) shall not include any RTE files.⌋ (SRS_E2E_08528)

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 is 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 OxFF).

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

Requirement	Description	Satisfied by
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_E2E_00032, SWS_E2E_00327, SWS_E2E_00467
SRS_BSW_00004	All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files	SWS_E2E_00038
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_E2E_00037
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_E2E_00037
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	SWS_E2E_00037
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_E2E_00294
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_E2E_00037
SRS_BSW_00171	Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time	SWS_E2E_00037
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_E2E_00047
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_E2E_00294
SRS_BSW_00337	Classification of development errors	SWS_E2E_00047
SRS_BSW_00338	-	SWS_E2E_00049, SWS_E2E_00294
SRS_BSW_00339	Reporting of production relevant error status	SWS_E2E_00216, SWS_E2E_00294

SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_E2E_00037
SRS_BSW_00345	BSW Modules shall support pre-compile configuration	SWS_E2E_00037
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_E2E_00049, SWS_E2E_00294
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_E2E_00294
SRS_BSW_00435	-	SWS_E2E_00294
SRS_E2E_08527	E2E library shall provide E2E profiles, in a form of library functions	SWS_E2E_00050, SWS_E2E_00097, UC_E2E_00304, UC_E2E_00317, UC_E2E_00328
SRS_E2E_08528	E2E library shall provide E2E profiles, where each E2E profile completely defines a particular safety protocol	SWS_E2E_00011, SWS_E2E_00012, SWS_E2E_00017, SWS_E2E_00018, SWS_E2E_00020, SWS_E2E_00021, SWS_E2E_00033, SWS_E2E_00048, SWS_E2E_00075, SWS_E2E_00076, SWS_E2E_00085, SWS_E2E_00091, SWS_E2E_00092, SWS_E2E_00094, SWS_E2E_00095, SWS_E2E_00096, SWS_E2E_00098, SWS_E2E_00099, SWS_E2E_00106, SWS_E2E_00107, SWS_E2E_00110, SWS_E2E_00115, SWS_E2E_00118, SWS_E2E_00119, SWS_E2E_00120, SWS_E2E_00121, SWS_E2E_00122, SWS_E2E_00123, SWS_E2E_00124, SWS_E2E_00125, SWS_E2E_00126, SWS_E2E_00127, SWS_E2E_00128, SWS_E2E_00129, SWS_E2E_00130, SWS_E2E_00132, SWS_E2E_00133, SWS_E2E_00134, SWS_E2E_00135, SWS_E2E_00136, SWS_E2E_00137, SWS_E2E_00138, SWS_E2E_00139, SWS_E2E_00140, SWS_E2E_00141, SWS_E2E_00142, SWS_E2E_00143, SWS_E2E_00145, SWS_E2E_00146, SWS_E2E_00147, SWS_E2E_00148, SWS_E2E_00149, SWS_E2E_00150, SWS_E2E_00151, SWS_E2E_00152, SWS_E2E_00153, SWS_E2E_00154, SWS_E2E_00158, SWS_E2E_00160, SWS_E2E_00161, SWS_E2E_00163, SWS_E2E_00166, SWS_E2E_00169, SWS_E2E_00190, SWS_E2E_00195, SWS_E2E_00196, SWS_E2E_00200, SWS_E2E_00215, SWS_E2E_00217, SWS_E2E_00227, SWS_E2E_00228, SWS_E2E_00276, SWS_E2E_00298, SWS_E2E_00299, SWS_E2E_00300, SWS_E2E_00301, SWS_E2E_00306, SWS_E2E_00307, SWS_E2E_00311, SWS_E2E_00314, SWS_E2E_00318, SWS_E2E_00319,

		SWS_E2E_00320, SWS_E2E_00321, SWS_E2E_00322, SWS_E2E_00323, SWS_E2E_00324, SWS_E2E_00325, SWS_E2E_00379, SWS_E2E_00380, SWS_E2E_00381, SWS_E2E_00382, SWS_E2E_00383, SWS_E2E_00384, SWS_E2E_00385, SWS_E2E_00386, SWS_E2E_00387, SWS_E2E_00388, SWS_E2E_00389, SWS_E2E_00390, SWS_E2E_00391, SWS_E2E_00392, SWS_E2E_00476, SWS_E2E_00477, UC_E2E_00051, UC_E2E_00053, UC_E2E_00055, UC_E2E_00057, UC_E2E_00061, UC_E2E_00062, UC_E2E_00063, UC_E2E_00071, UC_E2E_00072, UC_E2E_00073, UC_E2E_00087, UC_E2E_00089, UC_E2E_00165, UC_E2E_00170, UC_E2E_00173, UC_E2E_00192, UC_E2E_00202, UC_E2E_00203, UC_E2E_00204, UC_E2E_00205, UC_E2E_00206, UC_E2E_00207, UC_E2E_00208, UC_E2E_00209, UC_E2E_00213, UC_E2E_00224, UC_E2E_00226, UC_E2E_00230, UC_E2E_00232, UC_E2E_00233, UC_E2E_00235, UC_E2E_00237, UC_E2E_00239, UC_E2E_00240, UC_E2E_00241, UC_E2E_00242, UC_E2E_00248, UC_E2E_00249, UC_E2E_00250, UC_E2E_00251, UC_E2E_00255, UC_E2E_00256, UC_E2E_00257, UC_E2E_00258, UC_E2E_00259, UC_E2E_00261, UC_E2E_00262, UC_E2E_00263, UC_E2E_00264, UC_E2E_00265, UC_E2E_00266, UC_E2E_00267, UC_E2E_00268, UC_E2E_00270, UC_E2E_00271, UC_E2E_00272, UC_E2E_00273, UC_E2E_00274, UC_E2E_00275, UC_E2E_00277, UC_E2E_00278, UC_E2E_00279, UC_E2E_00280, UC_E2E_00288, UC_E2E_00289, UC_E2E_00290, UC_E2E_00292, UC_E2E_00293, UC_E2E_00296, UC_E2E_00297, UC_E2E_00300, UC_E2E_00301, UC_E2E_00308, UC_E2E_00313, UC_E2E_00315, UC_E2E_00320, UC_E2E_00326, UC_E2E_00465
SRS_E2E_08529	Each of the defined E2E profiles shall use an appropriate subset of specific protection mechanisms	SWS_E2E_00083, SWS_E2E_00218, SWS_E2E_00219, SWS_E2E_00372, SWS_E2E_00394, SWS_E2E_00479
SRS_E2E_08530	Each E2E profile shall have a unique ID, define precisely a set of mechanisms and its behavior in a semi-formal way	SWS_E2E_00196
SRS_E2E_08531	E2E library shall call the CRC routines of CRC library	SWS_E2E_00070, SWS_E2E_00117, SWS_E2E_00221, SWS_E2E_00329, SWS_E2E_00400, SWS_E2E_00420
SRS_E2E_08533	CRC used in a E2E profile shall be different than the CRC used by the underlying physical communication protocol	SWS_E2E_00083, SWS_E2E_00218, SWS_E2E_00219, SWS_E2E_00372, SWS_E2E_00394, SWS_E2E_00479
SRS_E2E_08534	E2E library shall provide	SWS_E2E_00022, SWS_E2E_00047,

	separate error flags and error counters for each type of detected communication failure	SWS_E2E_00214, SWS_E2E_00337, SWS_E2E_00437, SWS_E2E_00441
SRS_E2E_08536	Either SW-C or E2E Library shall compute the intermediate CRC over application data element [Approved]	SWS_E2E_00082, SWS_E2E_00126, SWS_E2E_00134, SWS_E2E_00330, SWS_E2E_00401, SWS_E2E_00421
SRS_E2E_08537	When using E2E Profiles 1/2, SW-Cs shall tolerate at least one received data element that is invalid/corrupted but not detected by E2E	UC_E2E_00170
SRS_E2E_08539	-	SWS_E2E_00326, SWS_E2E_00329, SWS_E2E_00334, SWS_E2E_00335, SWS_E2E_00336, SWS_E2E_00338, SWS_E2E_00339, SWS_E2E_00340, SWS_E2E_00342, SWS_E2E_00343, SWS_E2E_00344, SWS_E2E_00345, SWS_E2E_00347, SWS_E2E_00349, SWS_E2E_00350, SWS_E2E_00351, SWS_E2E_00352, SWS_E2E_00353, SWS_E2E_00354, SWS_E2E_00355, SWS_E2E_00356, SWS_E2E_00357, SWS_E2E_00358, SWS_E2E_00359, SWS_E2E_00360, SWS_E2E_00361, SWS_E2E_00362, SWS_E2E_00363, SWS_E2E_00364, SWS_E2E_00365, SWS_E2E_00366, SWS_E2E_00367, SWS_E2E_00368, SWS_E2E_00369, SWS_E2E_00370, SWS_E2E_00371, SWS_E2E_00373, SWS_E2E_00375, SWS_E2E_00376, SWS_E2E_00377, SWS_E2E_00378, SWS_E2E_00393, SWS_E2E_00397, SWS_E2E_00399, SWS_E2E_00400, SWS_E2E_00401, SWS_E2E_00403, SWS_E2E_00404, SWS_E2E_00405, SWS_E2E_00406, SWS_E2E_00407, SWS_E2E_00409, SWS_E2E_00411, SWS_E2E_00412, SWS_E2E_00413, SWS_E2E_00414, SWS_E2E_00416, SWS_E2E_00417, SWS_E2E_00419, SWS_E2E_00420, SWS_E2E_00421, SWS_E2E_00423, SWS_E2E_00424, SWS_E2E_00425, SWS_E2E_00426, SWS_E2E_00427, SWS_E2E_00428, SWS_E2E_00429, SWS_E2E_00430, SWS_E2E_00431, SWS_E2E_00432, SWS_E2E_00433, SWS_E2E_00434, SWS_E2E_00436, SWS_E2E_00437, SWS_E2E_00438, SWS_E2E_00439, SWS_E2E_00440, SWS_E2E_00441, SWS_E2E_00443, SWS_E2E_00444, SWS_E2E_00445, SWS_E2E_00446, SWS_E2E_00447, SWS_E2E_00448, SWS_E2E_00449,

		SWS_E2E_00450, SWS_E2E_00451, SWS_E2E_00452, SWS_E2E_00453, SWS_E2E_00454, SWS_E2E_00455, SWS_E2E_00456, SWS_E2E_00457, SWS_E2E_00458, SWS_E2E_00459, SWS_E2E_00460, SWS_E2E_00461, SWS_E2E_00462, SWS_E2E_00466, SWS_E2E_00469, SWS_E2E_00470, SWS_E2E_00471, SWS_E2E_00472, SWS_E2E_00473, SWS_E2E_00478, UC_E2E_00236, UC_E2E_00316, UC_E2E_00327, UC_E2E_00463, UC_E2E_00464

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 an 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 works as follows:
- 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.

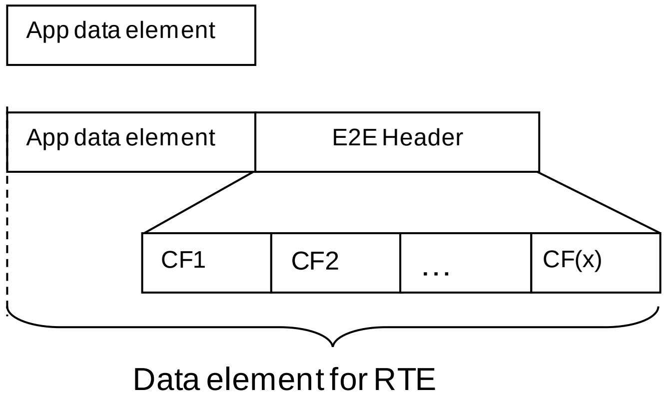


Figure 7-1: Safety protocol concept (with exemplary location of the E2E header)

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_E2E_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_E2E_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.⌋ (SRS_E2E_08528)

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

[SWS_E2E_00047]⌈ The following error flags for errors shall be used by all E2E Library functions:

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
Function executed in wrong state	Development error or integration error	E2E_E_WRONGSTATE	0x1A

⌋ (SRS_BSW_00337, SRS_BSW_00323, SRS_E2E_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_PXXProtect() / E2E_PXXCheckshall handle the errors/stati defined in SWS_E2E_00047 according to the column "How do caller of E2E shall handle it".⌋ (SRS_E2E_08528)

In other words, the E2E libary 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.⌋ (SRS_E2E_08528)

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_P01CheckStatusType.
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: • 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 (x8 + x4 + x3 + x2 + 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.

⌋ (SRS_E2E_08529, SRS_E2E_08533)

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 basis 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, which are using E2E-Library [2)] for a set of data protected by same CRC	

Table 7-1: Detectable communication faults using Profile 1

[SWS_E2E_00070]⌈ E2E Profile 1 shall use the polynomial of CRC-8-SAE J1850, i.e. the polynomial 0x1D (x8 + x4 + x3 + x2 + 1), but with start value and XOR value equal to 0x00.⌋ (SRS_E2E_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.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

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)⌋ (SRS_E2E_08528)

[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).⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08529, SRS_E2E_08533)

[SWS_E2E_00190]⌈ E2E Profile 1 shall use the Crc_CalculateCRC8 () function of the SWS CRC Library for calculating CRC checksums.⌋ (SRS_E2E_08528)

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:

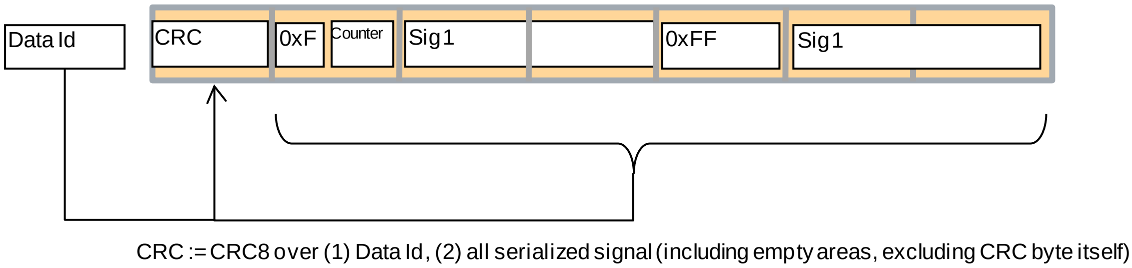


Figure 7-2: E2E Profile 1 variant 1A CRC calculation example

For standard variant 1C:

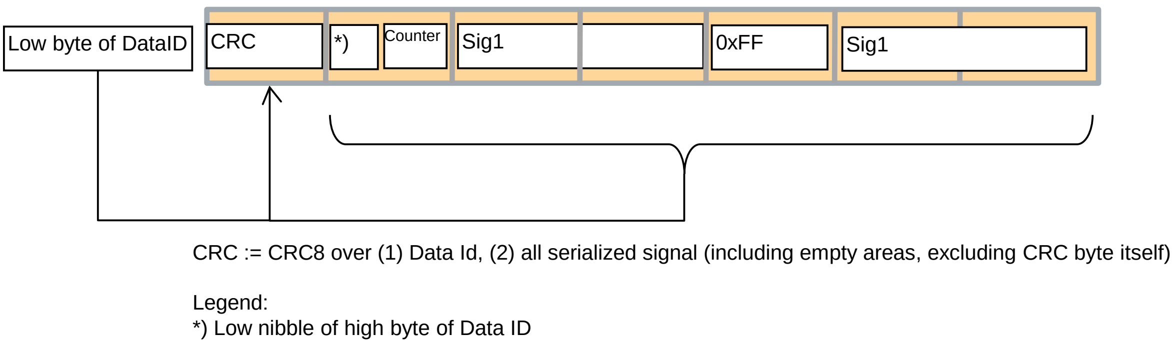


Figure 7-3: E2E Profile 1 variant 1C CRC calculation example

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

Over all transmitted bytes of a safety-related complex data element/signal group (except the CRC byte).⌋ (SRS_E2E_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. SignalIPdu.unusedBitPattern = 0xFF.⌋ (SRS_E2E_08528)

[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. SignalIPdu.unusedBitPattern = 0xFF.⌋ (SRS_E2E_08528)

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

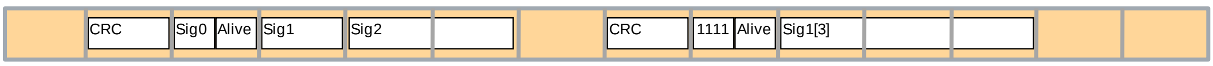


Figure 7-4: E2E Profile 1 example layout (two signal groups protected by E2E in one I-PDU)

[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. SignalIPdu.unusedBitPattern = 0xFF.⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08528)

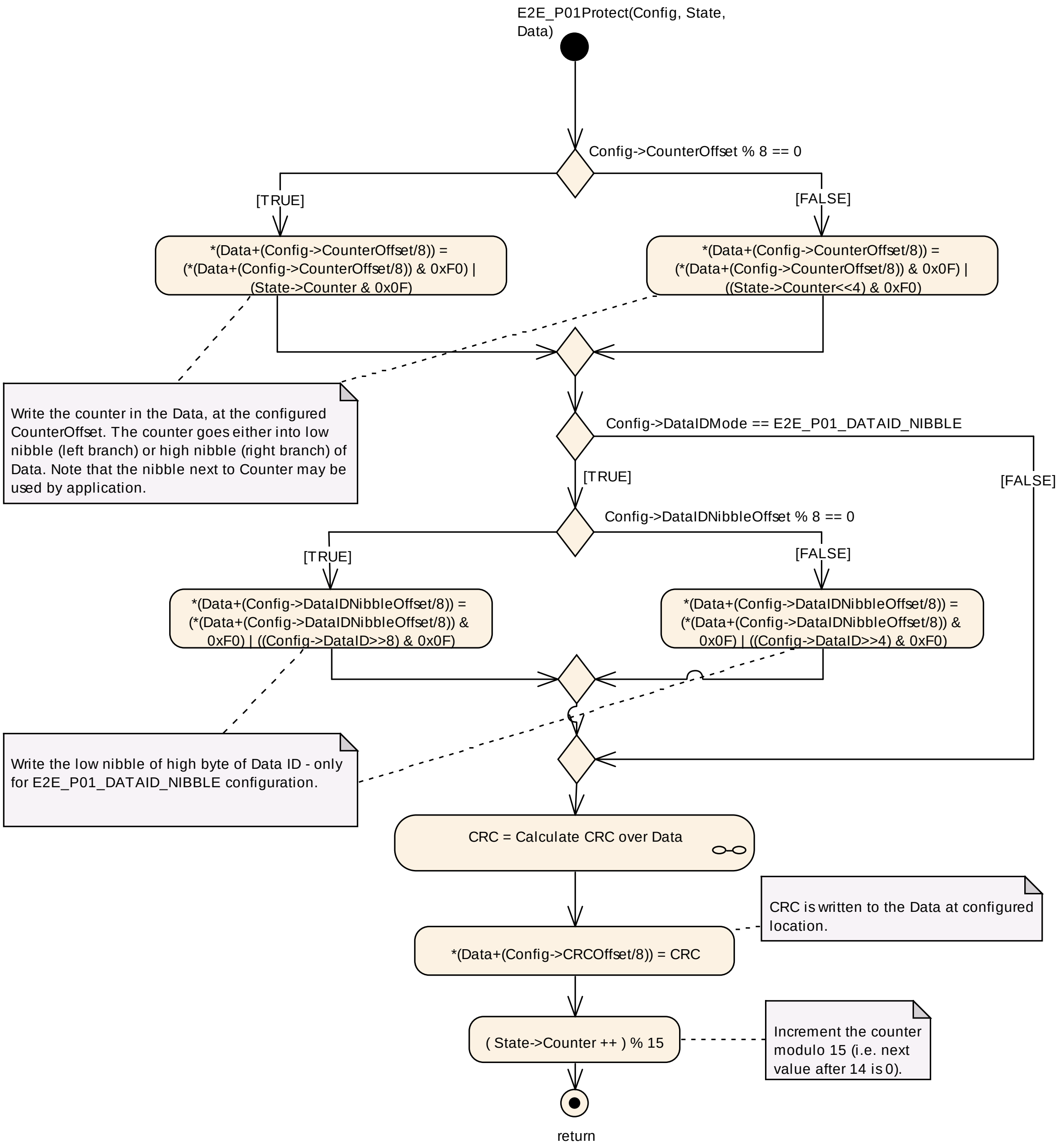


Figure 7-5: E2E_P01Protect()

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:

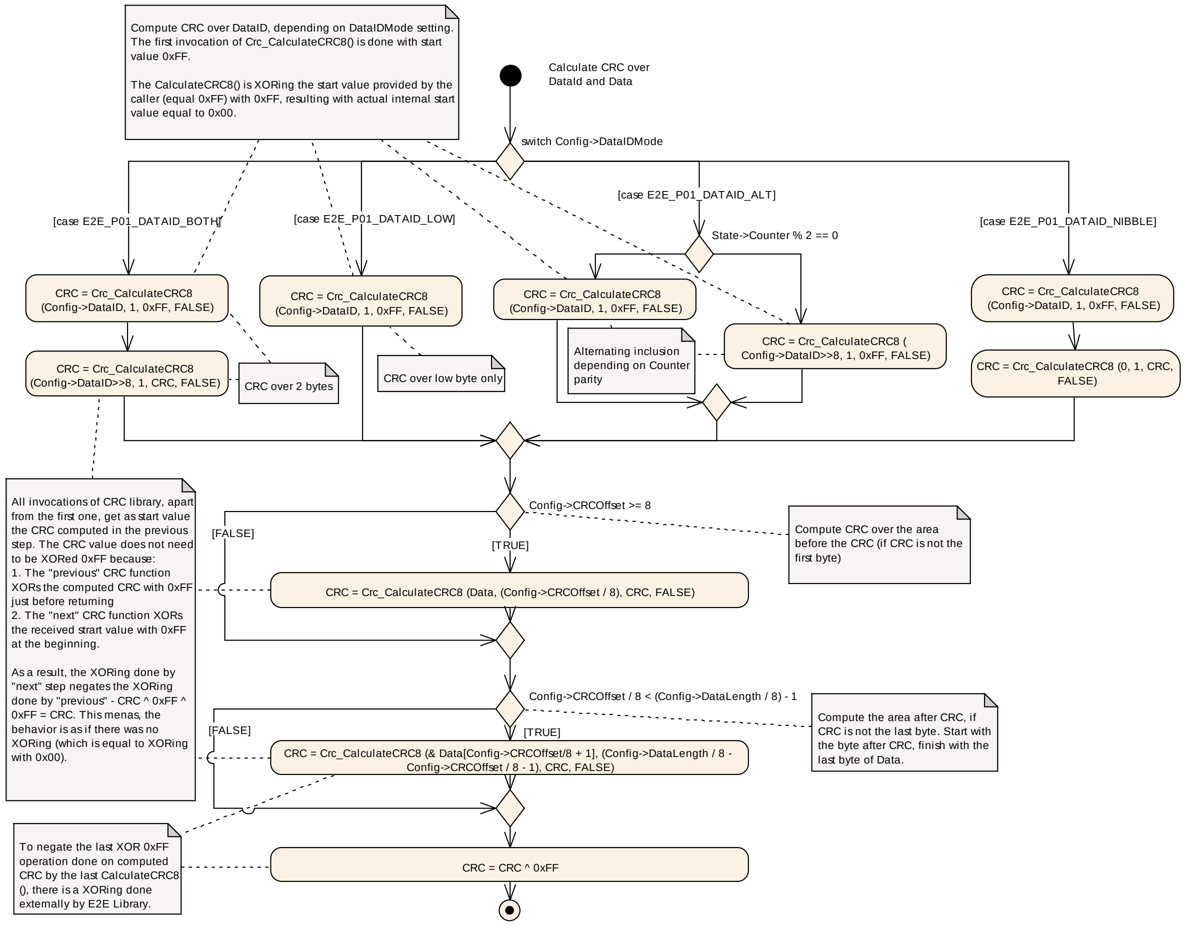


Figure 7-6: Subdiagram "Calculate CRC over Data ID and Data", used by E2E_P01Protect() and E2E_P01Check()

It is important to note that the function Crc_CalculateCRC8 of CRC library / CRC routines have changed is 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 OxFF).

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.⌋ (SRS_E2E_08528, SRS_E2E_08530)

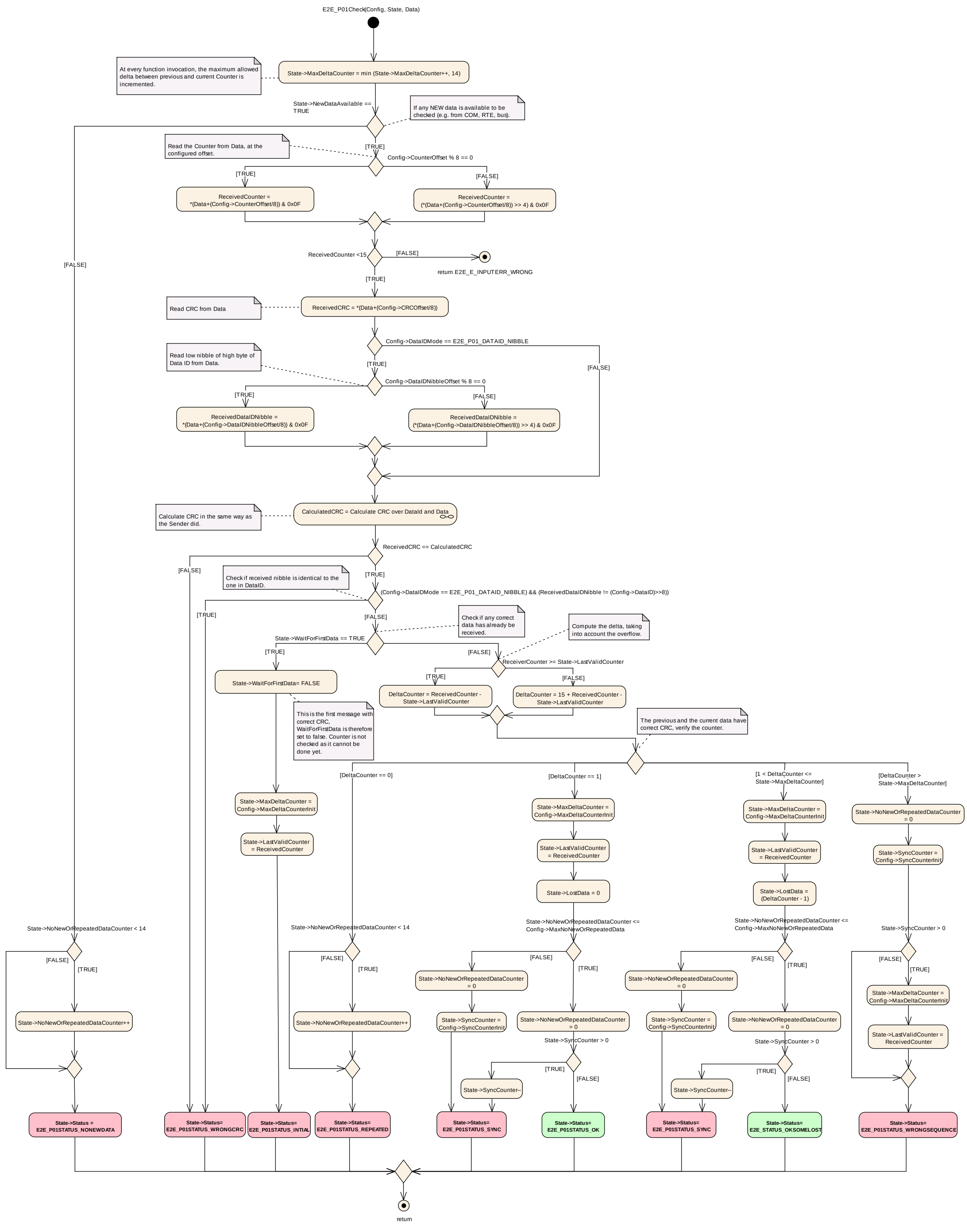


Figure 7-7: E2E_P01Check()

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 an 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 (x8 + x5 + x3 + x2 + x + 1) Start value: 0xFF Final XOR-value: 0xFF Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay and CAN.

⌋ (SRS_E2E_08529, SRS_E2E_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 7-2: 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_E2E_08531)

[SWS_E2E_00118]⌈ E2E Profile 2 shall use 0xFF as the start value CRC_StartValue8 for CRC calculation.⌋ (SRS_E2E_08528)

[SWS_E2E_00119]⌈ In E2E Profile 2, the specific Data ID used to calculate a specific CRC shall be of length 8 bit.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

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 below, with a maximum length of 256 bytes (i.e. N=255)

Data[0]	Data[1]		Data[2]	...	...	...	Data[N-1]	Data[N]
7 CRC 0	7 4	Counter	7 ... 0				7 ... 0	7 ... 0

⌋ (SRS_E2E_08528)

[SWS_E2E_00122]⌈ In E2E Profile 2, the CRC shall be Data[0].⌋ (SRS_E2E_08528)

[SWS_E2E_00123]⌈ In E2E Profile 2, the Counter shall be the low nibble (Bit 0…Bit 3) of Data[1].⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

[SWS_E2E_00125]⌈ In E2E Profile 2, the E2E_P02Check() function shall not modify any bit in Data.⌋ (SRS_E2E_08528)

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 7-2. 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 7-8 and Figure 7-9.

[SWS_E2E_00126]⌈ In E2E Profile 2, the E2E_P02Protect() function shall perform the activities as specified in Figure 7-8 and Figure 7-9.⌋ (SRS_E2E_08528, SRS_E2E_08536)

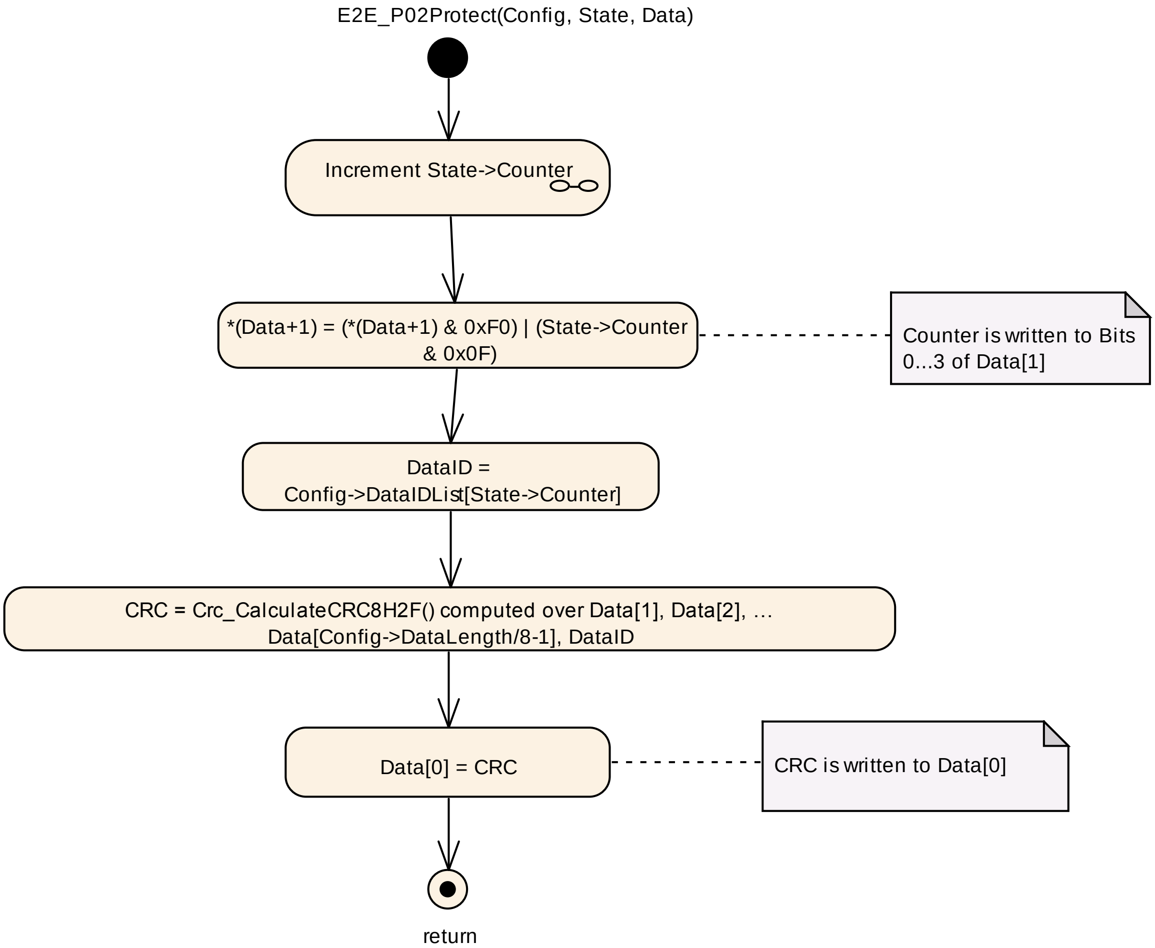


Figure 7-8: E2E_P02Protect()

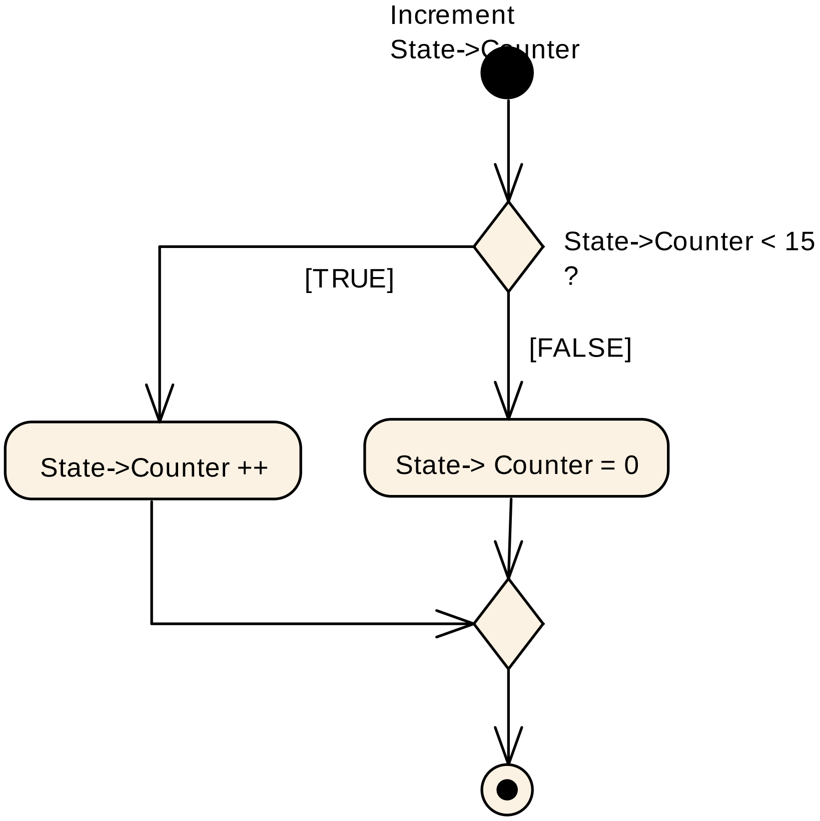


Figure 7-9: Increment Counter

[SWS_E2E_00127]⌈ In E2E Profile 2, the E2E_P02Protect() function shall increment the Counter of the state (E2E_P02ProtectStateType) by 1 on every transmission request from the sending SW-C, i.e. on every call of E2E_P02Protect().⌋ (SRS_E2E_08528)

[SWS_E2E_00128]⌈ In E2E Profile 2, the range of the value of the Counter shall be [0…15].⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

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.⌋
(SRS_E2E_08528)

[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.⌋
(SRS_E2E_08528)

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 7-10 Figure 7-11 and Figure 7-12.
[SWS_E2E_00134]⌈ In E2E Profile 2, the E2E_P02Check() function shall perform the activities as specified in Figure 7-10, Figure 7-11and Figure 7-12.⌋
(SRS_E2E_08528, SRS_E2E_08536)

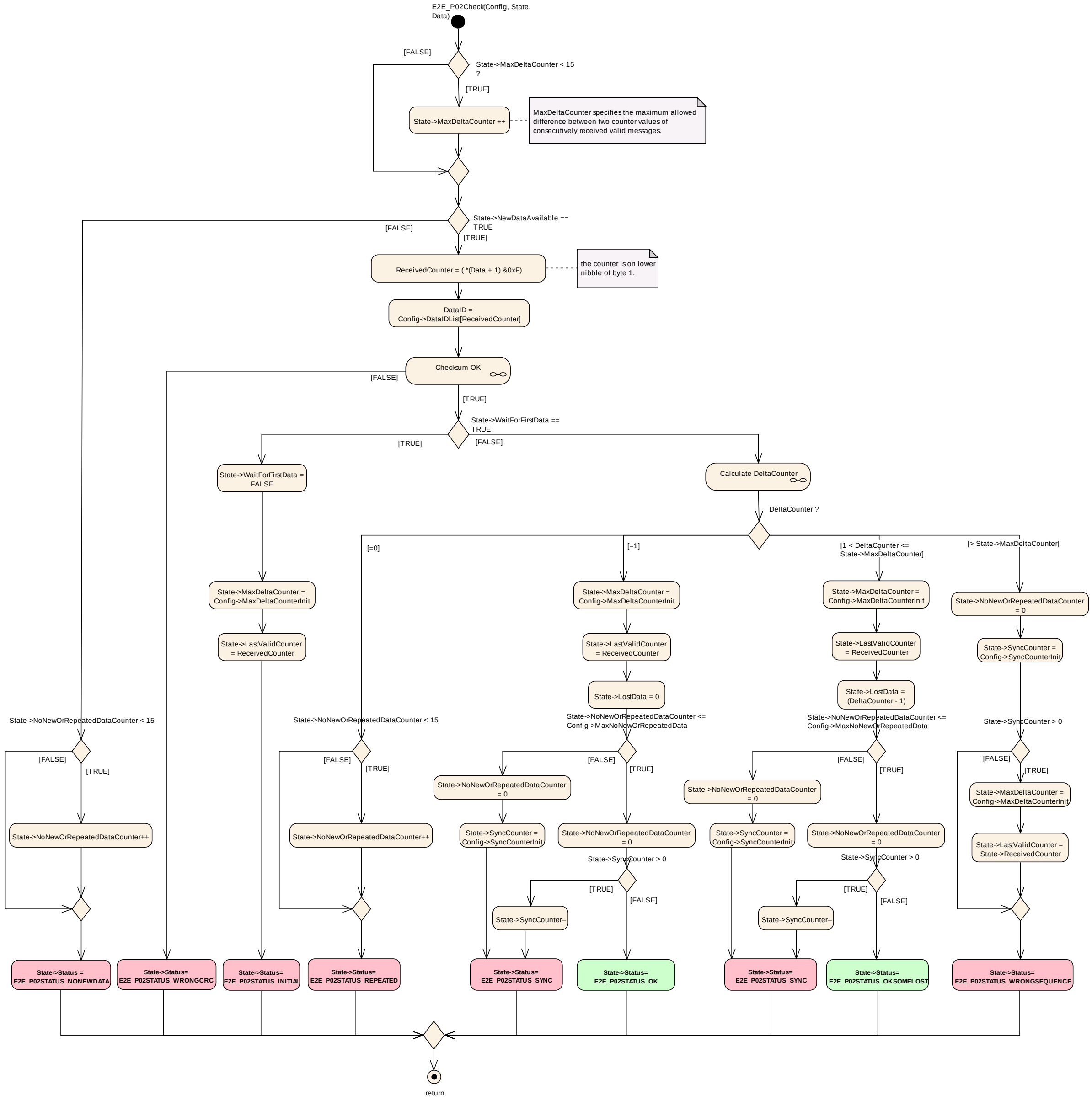


Figure 7-10: E2E_P02Check()

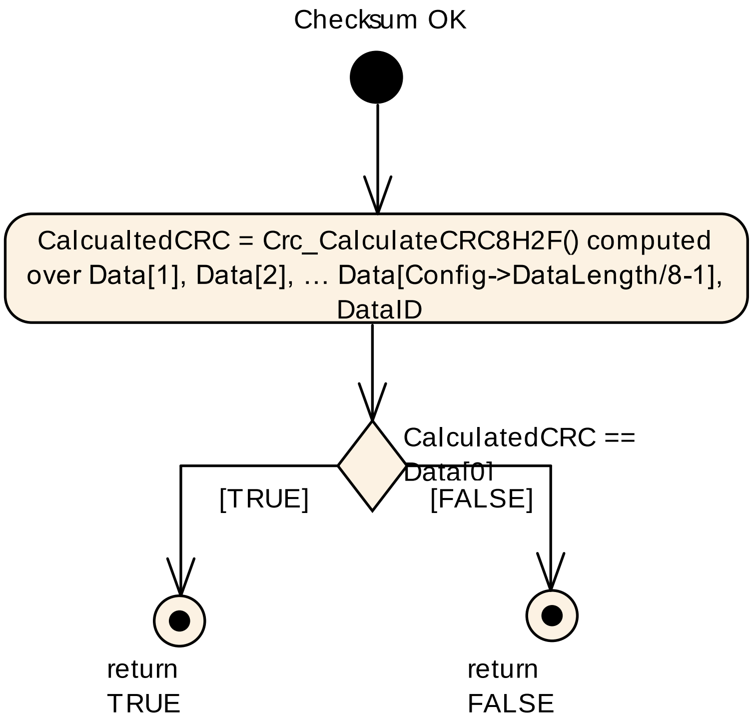


Figure 7-11: Checksum OK

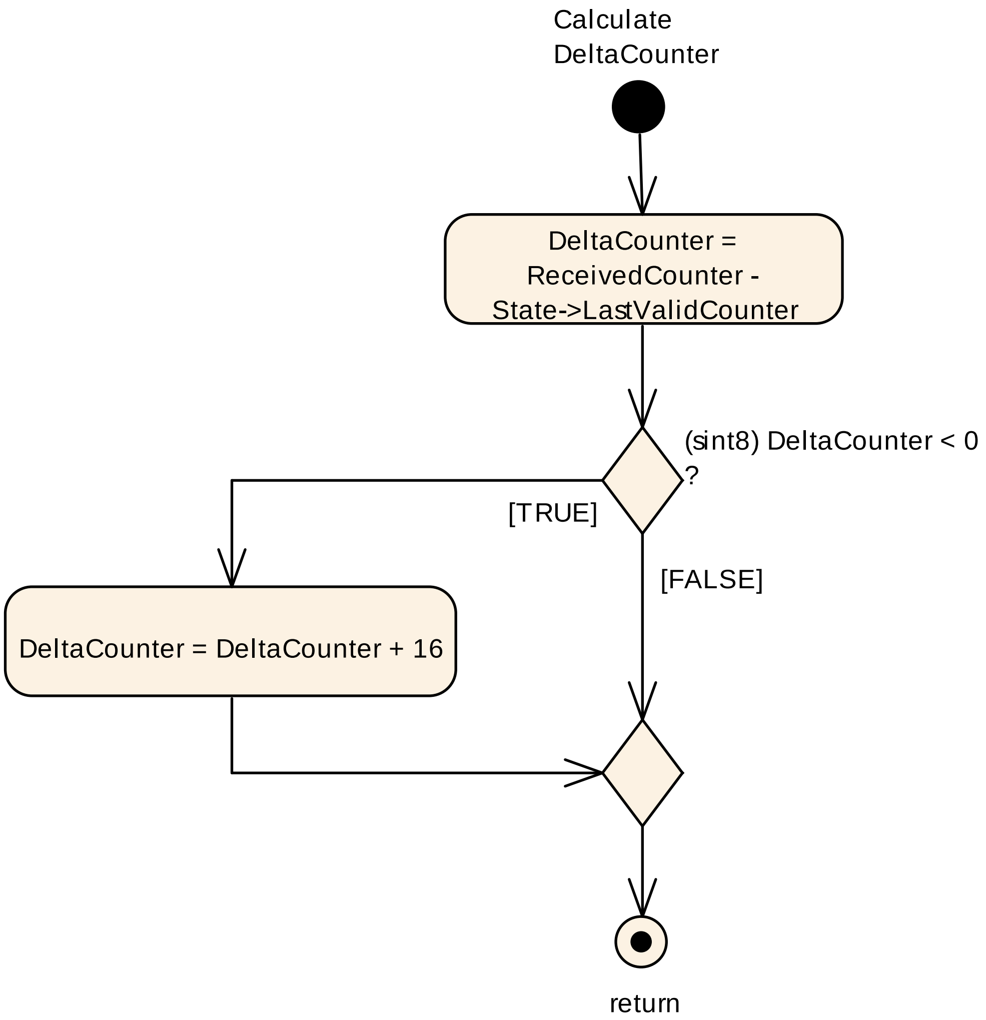


Figure 7-12: Calculate DeltaCounter

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].⌋ (SRS_E2E_08528)

Details on the calculation of DeltaCounter are depicted in Figure 7-12.
[SWS_E2E_00136]⌈ In E2E Profile 2, MaxDeltaCounter shall specify the maximum allowed difference between two Counter values of two consecutively received valid messages.⌋ (SRS_E2E_08528)

[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).⌋ (SRS_E2E_08528)

[SWS_E2E_00138]⌈ In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_NONEWDATA if the attribute NewDataAvailable is FALSE.⌋ (SRS_E2E_08528)
[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.
⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

[SWS_E2E_00142]⌈ In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_INITIAL if the flag WaitForFirstData is TRUE. ⌋ (SRS_E2E_08528)

[SWS_E2E_00143]⌈ In E2E Profile 2, the E2E_P02Check() function shall clear the flag WaitForFirstData if it returns with Status E2E_P02STATUS_INITIAL. ⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08528)

[SWS_E2E_00146]⌈ The E2E_P02Check() function shall set Status to E2E_P02STATUS_REPEATED if the calculated DeltaCounter equals 0. ⌋ (SRS_E2E_08528)

[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. State → NoNewOrRepeatedDataCounter ≤ Config → MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0.⌋ (SRS_E2E_08528)

[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 ≤ DeltaCounter ≤ MaxDeltaCounter); and

- the value of the NoNewOrRepeatedDataCounter exceeds MaxNoNewOrRepeatedData. (i.e. State → NoNewOrRepeatedDataCounter > Config → MaxNoNewOrRepeatedData)⌋ (SRS_E2E_08528)

[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 ≤ DeltaCounter ≤ MaxDeltaCounter); and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State → NoNewOrRepeatedDataCounter ≤ Config → MaxNoNewOrRepeatedData); and
- the SyncCounter exceeds 0.⌋ (SRS_E2E_08528)

[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 < DeltaCounter ≤ MaxDeltaCounter); and
- the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State → NoNewOrRepeatedDataCounter ≤ Config → MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

[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).⌋ (SRS_E2E_08528)

[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).⌋ (SRS_E2E_08528)

[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⌋ (SRS_E2E_08528)

[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).⌋ (SRS_E2E_08528)

7.5 Specification of E2E Profile 4

[SWS_E2E_00372]⌈ Profile 4 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Length	16 bits, to support dynamic-size data.
Counter	16-bits.
CRC	32 bits, polynomial in normal form 0x1F4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCP/IP.
Data ID	32-bits, unique system-wide.

⌋ (SRS_E2E_08529, SRS_E2E_08533)

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

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 7-3: Detectable communication faults using Profile 4

For details of CRC computation, the usage of start values and XOR values see CRC Library [7].

7.5.1 Data Layout

7.5.1.1 User data layout

In the E2E Profile 4, the user data layout (of the data to be protected) is not constrained by E2E Profile 4 – there is only a requirement that the length of data to be protected is multiple of 1 byte.

7.5.1.2 Header layout

The header of the E2E Profile 4 has one fixed layout, as follows:

Transmission order	0								1								2								3							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	E2E Length																E2E Counter															
4	E2E Data ID																															
8	E2E CRC																															

Figure 7-13: E2E Profile 4 header

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte fist) – imposed by profile
2. LSB Fist (least significant bit within byte first) – imposed by TCP/IP bus

For example, the 16 bits of the E2E counter are transmitted in the following order (higher number meaning higher significance):
7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7.

The header can be placed at a specific location in the protected data, by configuring the offset of the entire E2E header.

7.5.2 Counter

In E2E Profile 4, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E library.

[SWS_E2E_00478]⌈ In E2E Profile 4, 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. When the counter reaches the maximum value (0xFF'FF), then it shall restart with 0 for the next send request.⌋ (SRS_E2E_08539)

Note: This specification was previously falsely identified as SWS_E2E_00324.

Note that the counter value 0xFF'FF is not reseved as a special invalid value, but it is used as a normal counter value.

[SWS_E2E_00471]⌈ In E2E Profile 4, on the receiver side, the counter shall be initialized with 0xFF'FF.⌋ (SRS_E2E_08539)

In E2E Profile 4, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:
    a. no new data has arrived since last invocation of E2E library check function,
    b. the data is repeated
2. OK:
    a. counter is incremented by one (i.e. no data lost),
    b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),
3. Wrong sequence:
    a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

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

7.5.3 Data ID

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

[SWS_E2E_00326]⌈ In the E2E Profile 4, the Data ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.⌋ (SRS_E2E_08539)

There are currently no limitations on the values of Data ID – any values within the addres space of 32 bits are allowed.

[UC_E2E_00327]⌈ In the E2E profile 4, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data).⌋ (SRS_E2E_08539)

In case of usage of E2E Library for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Library using Data ID.

In case of usage of E2E Library for protecting I-PDUs (i.e. invocation from COM), the receiver COM espects at a reception only a specific I-PDU, which is checked by E2E Library using Data ID.

7.5.4 Length

The Length field is introduced to support variable-size length – the Data[] array storing the serialized data can potentially have a different length in each cycle.

7.5.5 CRC

E2E Profile 4 uses a 32-bit CRC, to ensure a high detection rate and high Hamming Distance.

[SWS_E2E_00329]⌈ E2E Profile 4 shall use the Crc_CalculateCRC32P4 () function of the SWS CRC Library for calculating the CRC. ⌋ (SRS_E2E_08539, SRS_E2E_08531)

Note: The CRC used by E2E Profile 4 is different from the CRCs used by FlexRay, CAN and TCP/IP. It is also provided by different software modules (FlexRay, CAN and TCP/IP stack CRCs/checksums are provided by hardware support in Communication Controllers or by communication stack software, but not by CRC library).

[SWS_E2E_00330]⌈ In E2E Profile 4, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes) and over the user data.⌋ (SRS_E2E_08536)

7.5.6 Timeout detection

The previously mentioned mechanisms (CRC, Counter, Data ID, Length) 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 I-PDUs, 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.

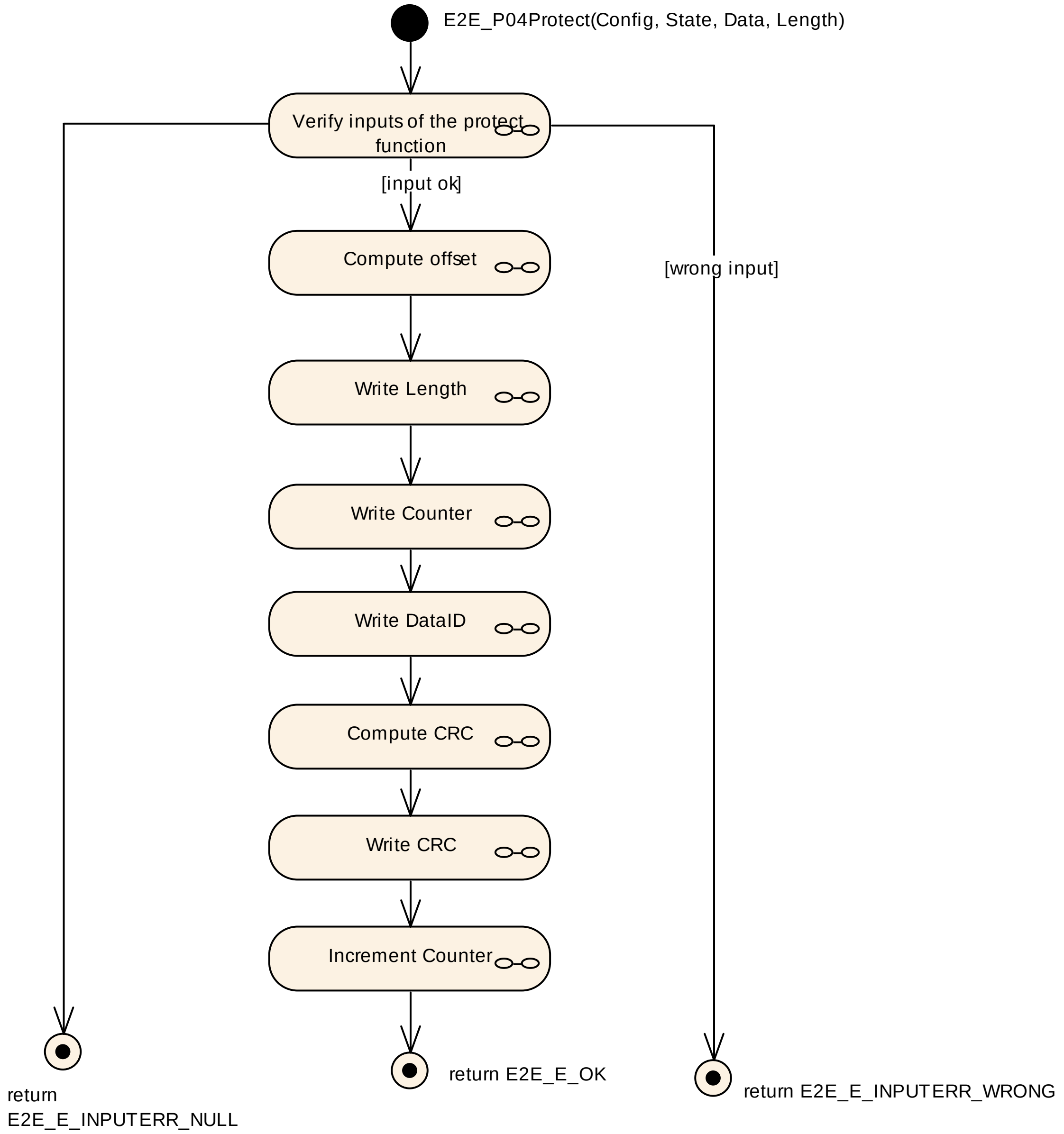
7.5.7 E2E Profile 4 variants

The E2E Profile 4 variants are specified in TPS System Specification.

7.5.8 E2E_P04Protect

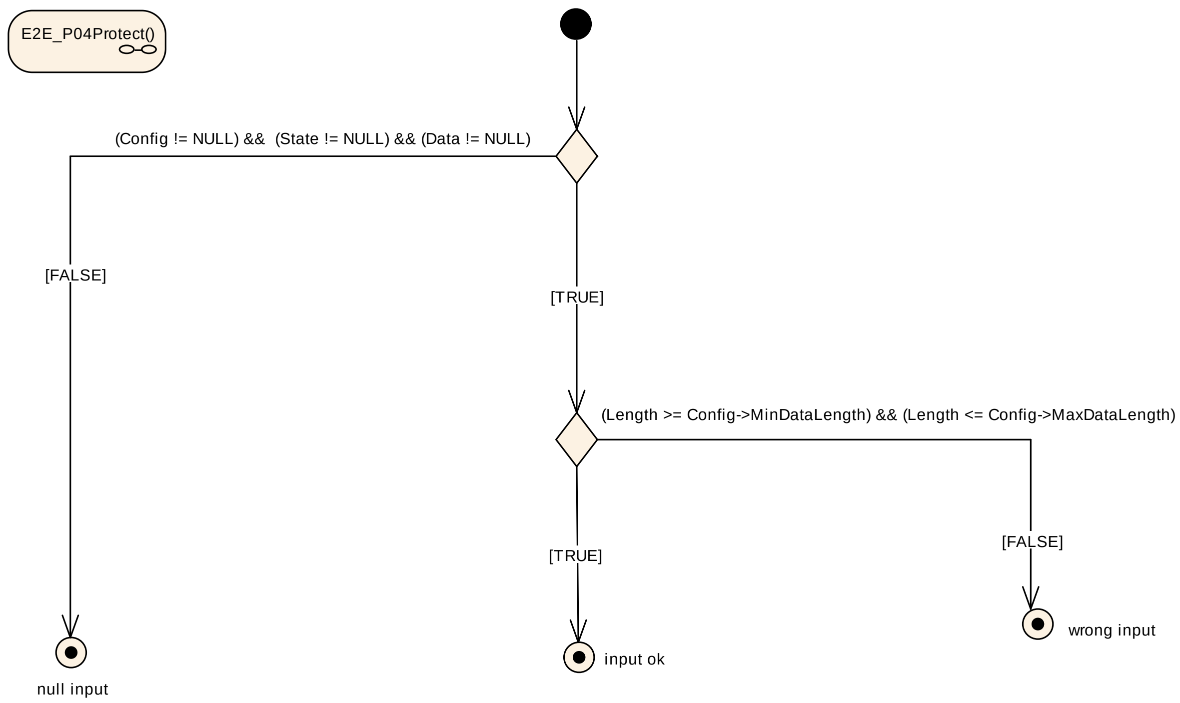
The function E2E_P04Protect() performs the steps as specified by the following eight diagrams in this section.

[SWS_E2E_00362]⌈ The function E2E_P04Protect() shall have the following overall behavior:



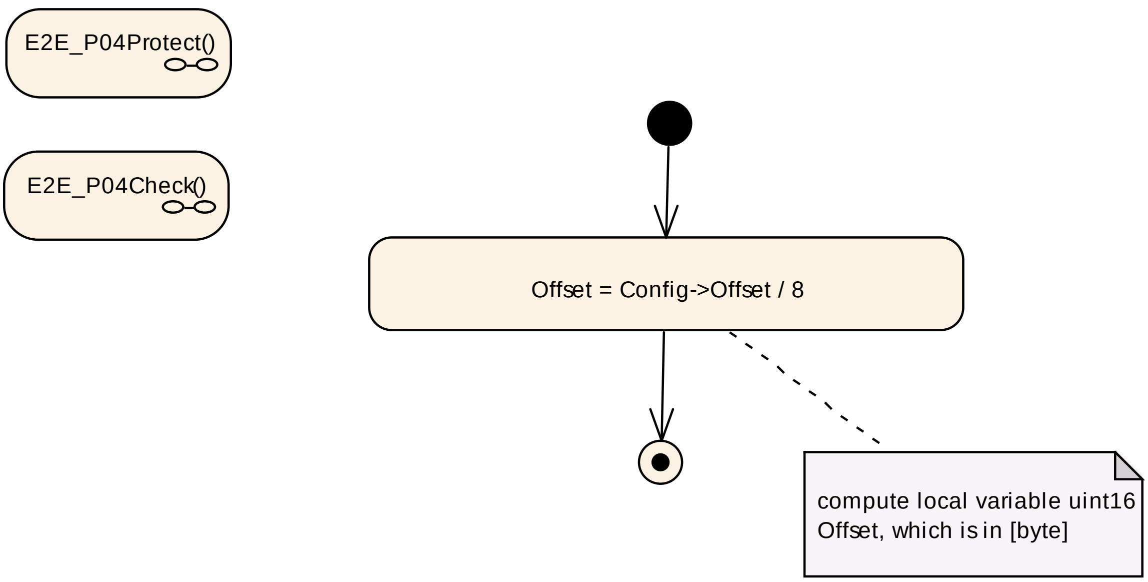
⌋ (SRS_E2E_08539)

[SWS_E2E_00363]⌈ The step "Verify inputs of the protect function" in E2E_P04Protect() shall have the following behavior:



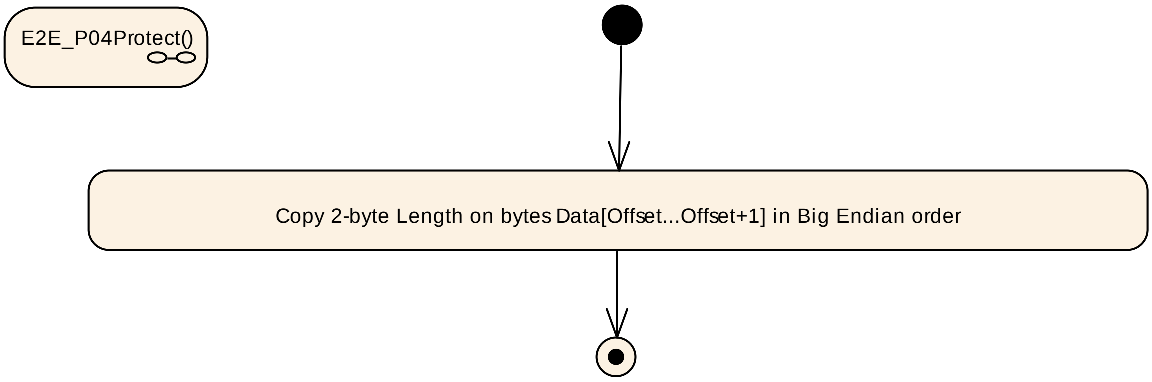
⌋ (SRS_E2E_08539)

[SWS_E2E_00376]⌈ The step "Compute offset" in E2E_P04Protect() and E2E_P04Check() shall have the following behavior:



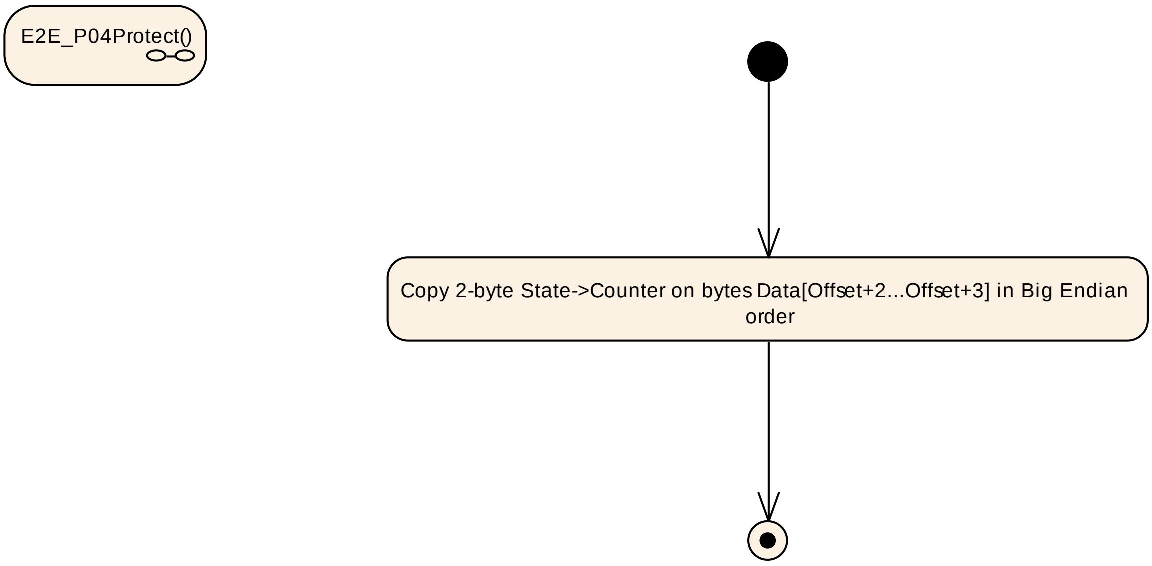
⌋ (SRS_E2E_08539)

[SWS_E2E_00364]⌈ The step "Write Length" in E2E_P04Protect() shall have the following behavior:



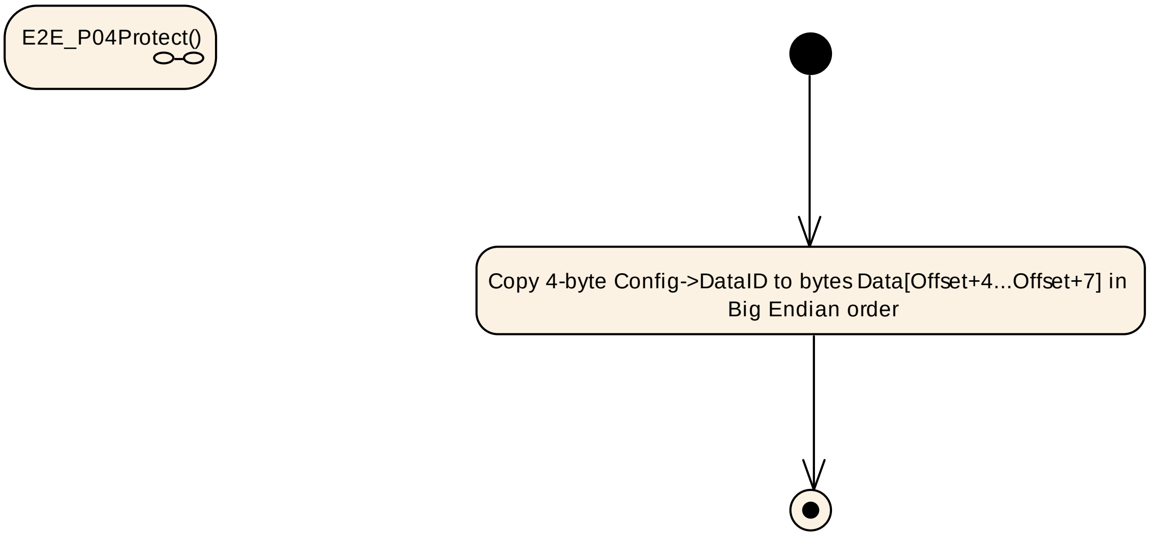
⌋ (SRS_E2E_08539)

[SWS_E2E_00365]⌈ The step “Write Counter” in E2E_P04Protect() shall have the following behavior:



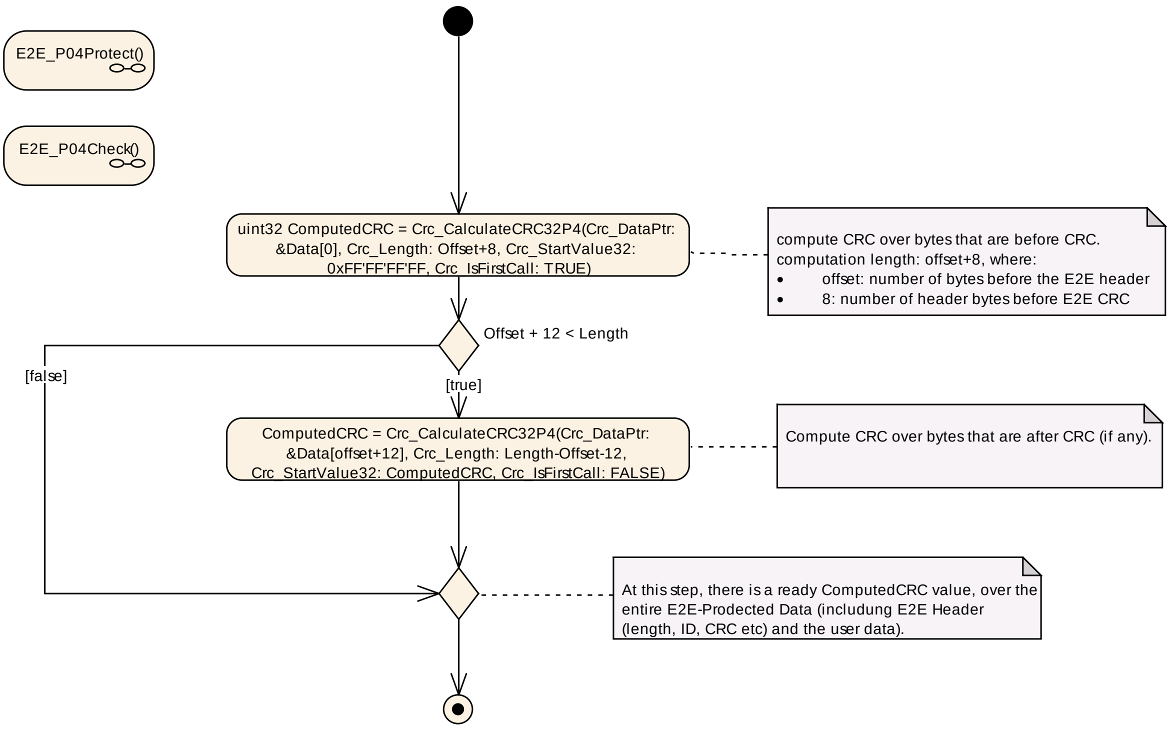
⌋ (SRS_E2E_08539)

[SWS_E2E_00366]⌈ The step “Write DataID” in E2E_P04Protect() shall have the following behavior:



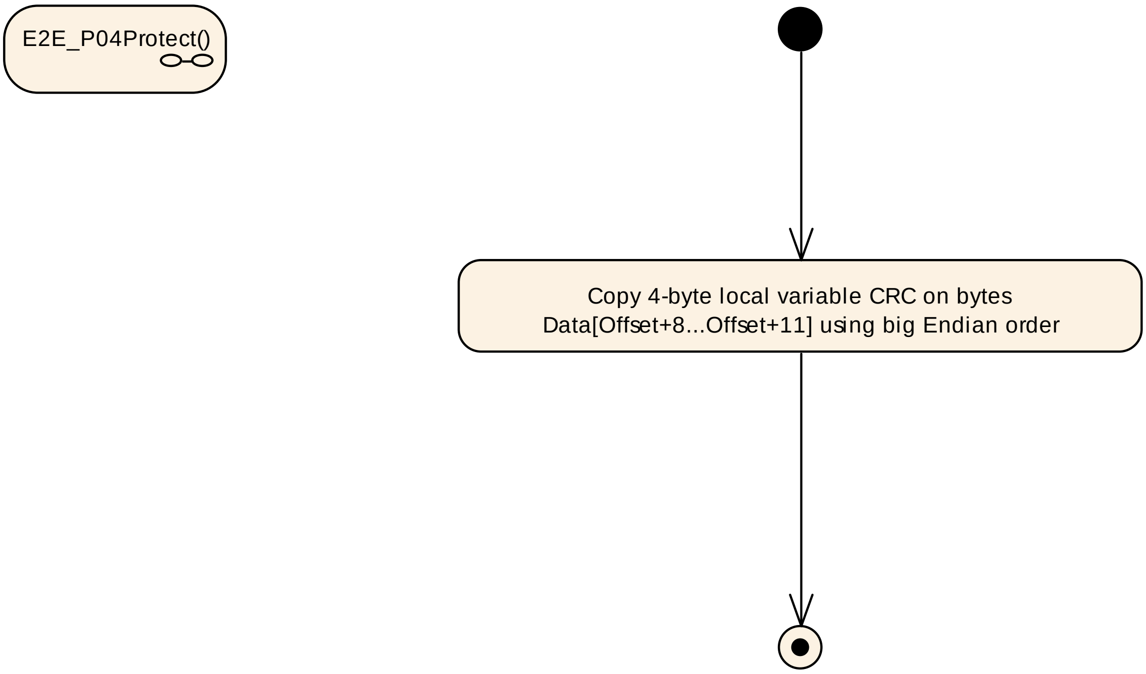
⌋ (SRS_E2E_08539)

[SWS_E2E_00367]⌈ The step "ComputeCRC" in E2E_P04Protect() and in E2E_P04Check() shall have the following behavior:



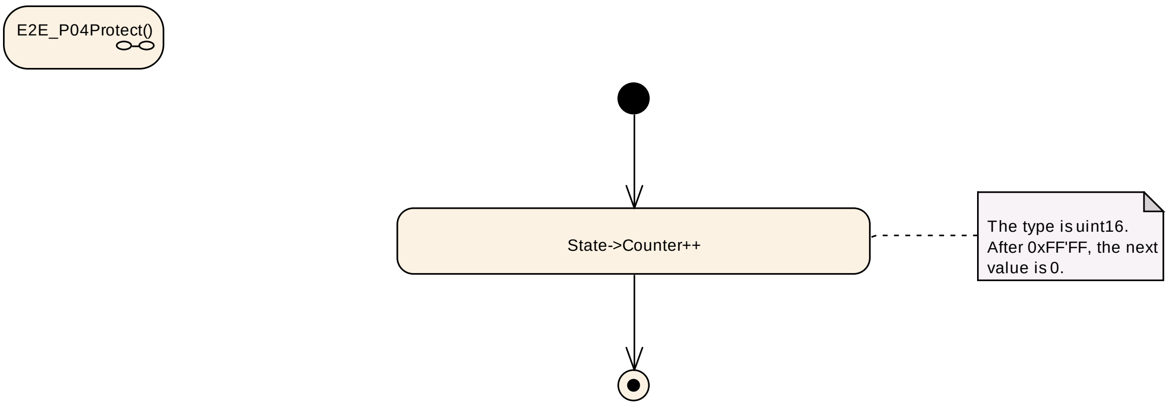
⌋ (SRS_E2E_08539)

[SWS_E2E_00368]⌈ The step "Write CRC" in E2E_P04Protect() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00369]⌈ The step “Increment Counter” in E2E_P04Protect() shall have the following behavior:

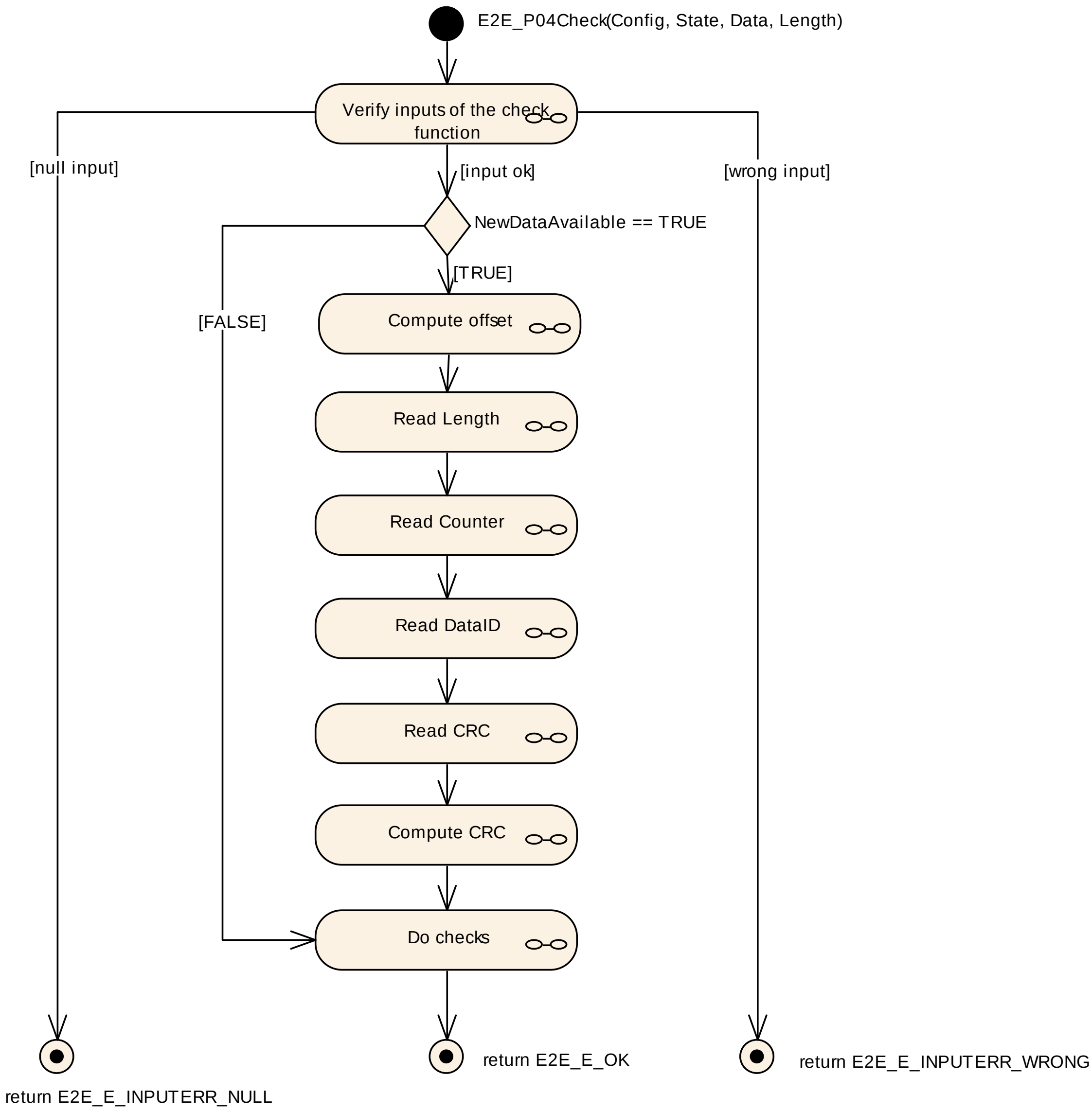


⌋ (SRS_E2E_08539)

7.5.9 E2E_P04Check

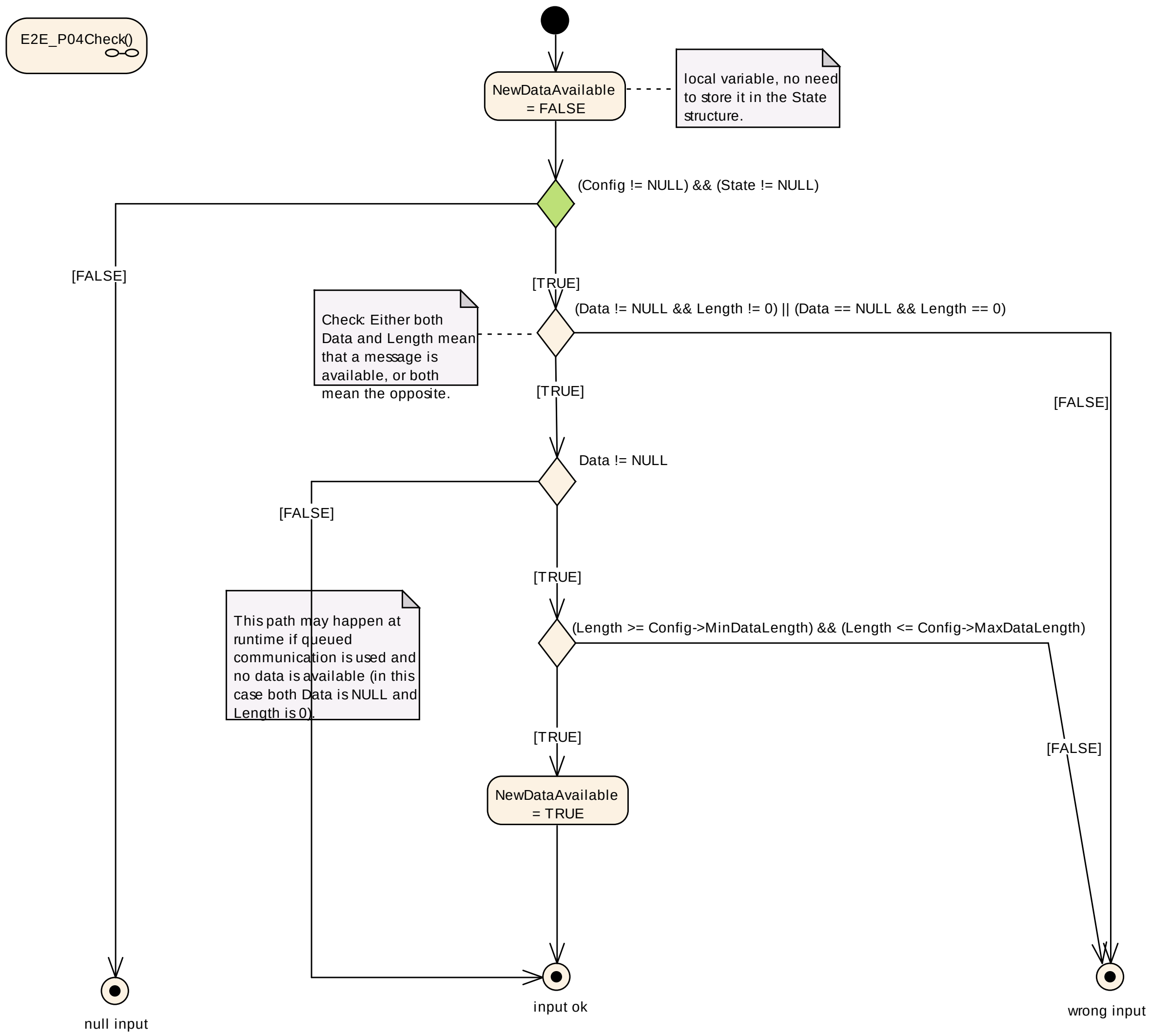
The function E2E_P04Check performs the actions as as specified by the following seven diagrams in this section and according to diagram SWS_E2E_00367.

[SWS_E2E_00355]⌈ The function E2E_P04Check() shall have the following overall behavior:



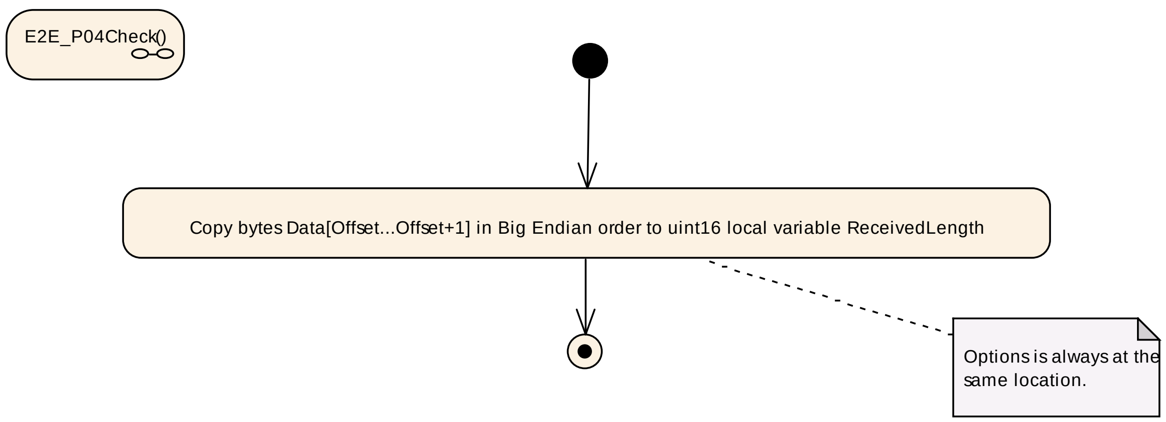
⌋ (SRS_E2E_08539)

[SWS_E2E_00356]⌈ The step "Verify inputs of the check function" in E2E_P04Check() shall have the following behavior:



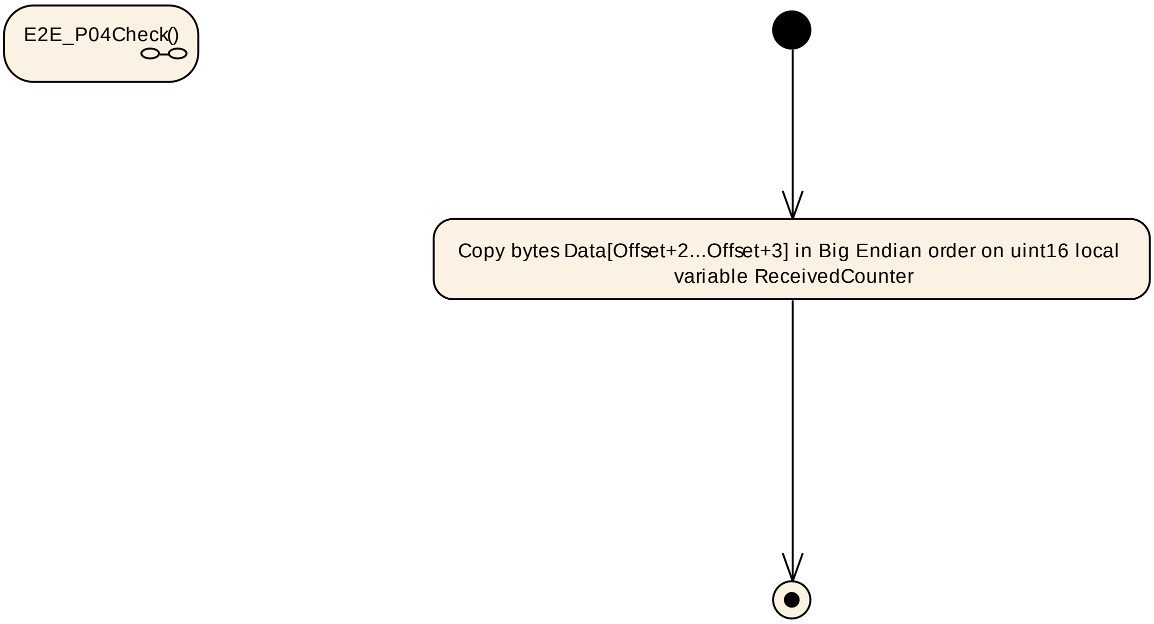
⌋ (SRS_E2E_08539)

[SWS_E2E_00357]⌈ The step “Read Length” in E2E_P04Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00358]⌈ The step “Read Counter” in E2E_P04Check() shall have the following behavior:

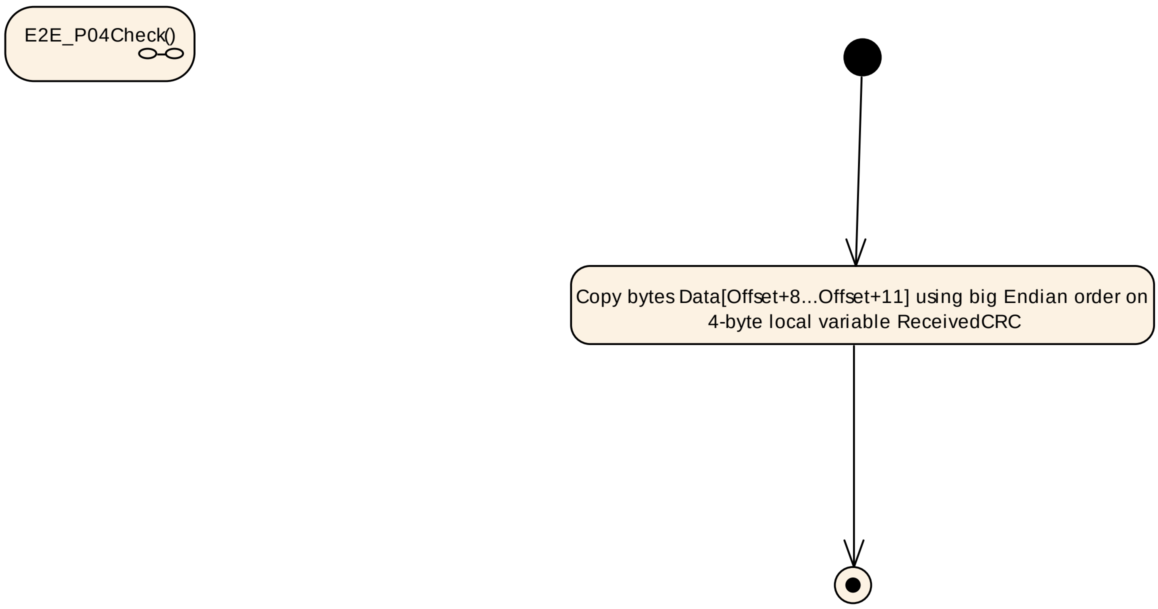


⌋ (SRS_E2E_08539)

[SWS_E2E_00359]⌈ The step “Read DataID” in E2E_P04Check() shall have the following behavior:

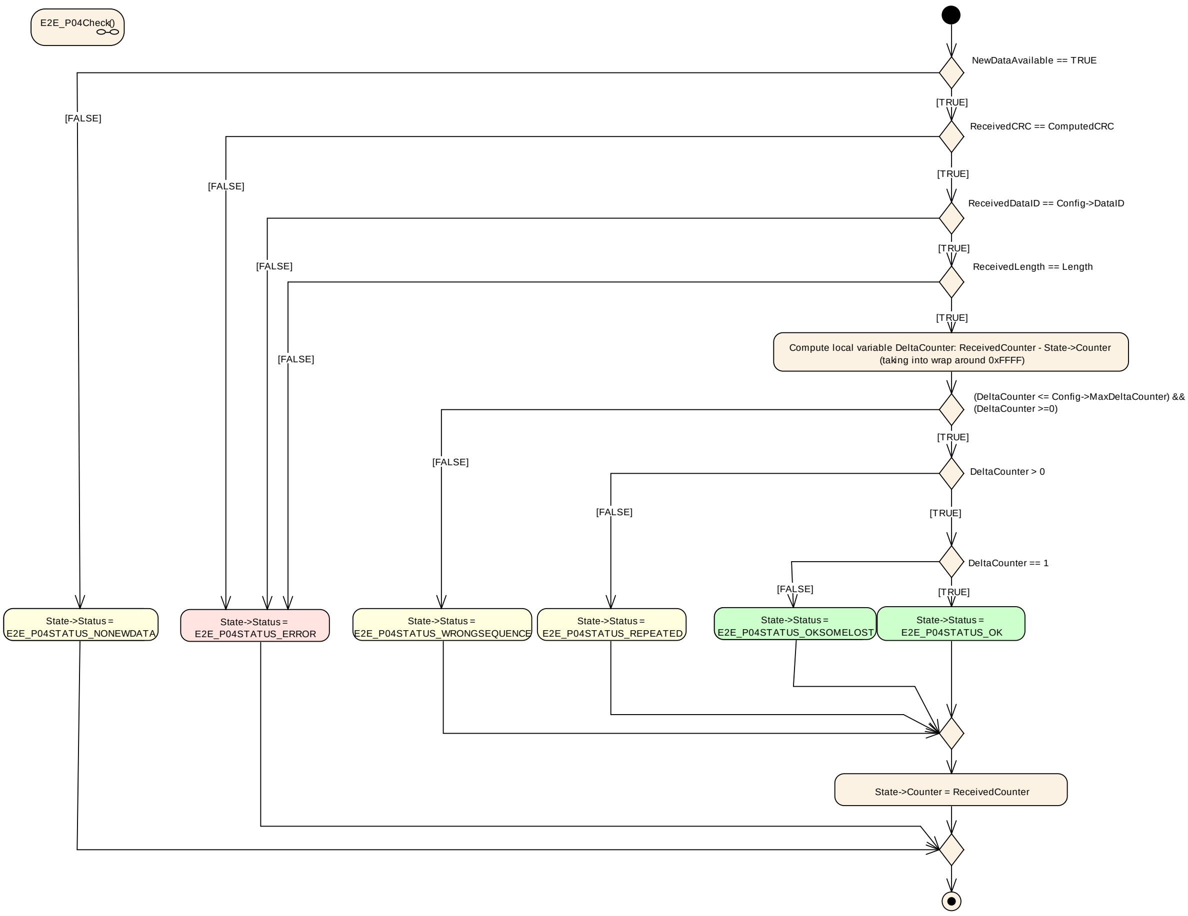
⌋ (SRS_E2E_08539)

[SWS_E2E_00360]⌈ The step “Read CRC” in E2E_P04Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00361]⌈ The step “Do Checks” in E2E_P04Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

7.6 Specification of E2E Profile 5

[SWS_E2E_00394]⌈ Profile 5 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Counter	8-bits. (explicitly sent)
CRC	16 bits, polynomial in normal form 0x1021 (Autosar notation), provided by CRC library. (explicitly sent)
Data ID	16-bits, unique system-wide. (implicitly sent)

⌋ (SRS_E2E_08529, SRS_E2E_08533)

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

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 7-4: Detectable communication faults using Profile 5

For details of CRC computation, the usage of start values and XOR values see CRC Library [7].

7.6.1 Data Layout

7.6.1.1 User data layout

In the E2E Profile 5, the user data layout (of the data to be protected) is not constrained by E2E Profile 5 – there is only a requirement, that the length of data to be protected is multiple of 1 byte.

7.6.1.2 Header layout

The header of the E2E Profile 5 has one fixed layout, as follows:

Transmission order	0								1								2							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	E2E CRC																E2E Counter							

Figure 7-14: E2E Profile 5 header

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

1. Little Endian (least significant byte fist) applicable for both implicit and explicit header fields – imposed by profile
2. MSB Fist (most significant bit within byte first) – imposed by Flexray/CAN bus.

7.6.2 Counter

In E2E Profile 5, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E library.

[SWS_E2E_00397]⌈ In E2E Profile 5, 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. When the counter reaches the maximum value (0xFF), then it shall restart with 0 for the next send request.⌋ (SRS_E2E_08539)

Note that the counter value 0xFF is not reserved as a special invalid value, but it is used as a normal counter value.

[SWS_E2E_00472]⌈ In E2E Profile 5, on the receiver side, the counter shall be initialized with 0xFF.⌋ (SRS_E2E_08539)

In E2E Profile 5, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:
    a. no new data has arrived since last invocation of E2E library check function,
    b. the data is repeated
2. OK:
    a. counter is incremented by one (i.e. no data lost),
    b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),
3. Error:
    a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

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

7.6.3 Data ID

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

[SWS_E2E_00399]⌈ In the E2E Profile 5, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation. ⌋ (SRS_E2E_08539)

The Data ID is not a part of the transmitted E2E header (similar to Profile 2 and 6).

[UC_E2E_00463]⌈ In the E2E profile 5, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data).
⌋ (SRS_E2E_08539)

In case of usage of E2E Library for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Library using Data ID.

In case of usage of E2E Library for protecting I-PDUs (i.e. invocation from COM), the receiver COM expects at a reception only a specific I-PDU, which is checked by E2E Library using Data ID.

7.6.4 Length

In Profile 5 there is no explicit transmission of the length.

7.6.5 CRC

E2E Profile 5 uses a 16-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[SWS_E2E_00400]⌈ E2E Profile 5 shall use the Crc_CalculateCRC16() function of the SWS CRC Library for calculating the CRC (Polynomial: 0x1021; Autosar notation).⌋ (SRS_E2E_08539, SRS_E2E_08531)

[SWS_E2E_00401]⌈ In E2E Profile 5, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended at the end with the Data ID.⌋ (SRS_E2E_08539, SRS_E2E_08536)

7.6.6 Timeout detection

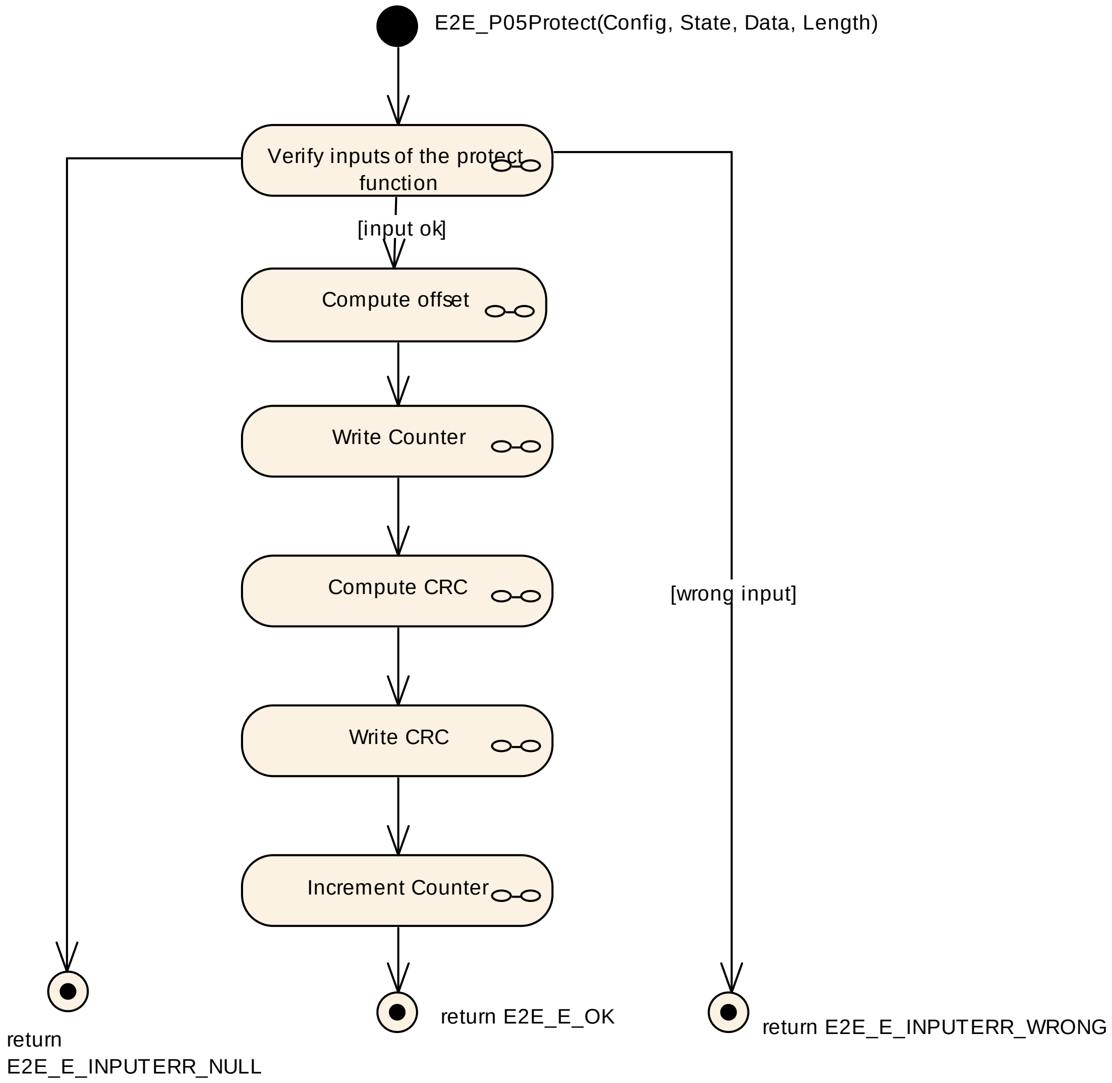
The previously mentioned mechanisms (for Profile 5: CRC, Counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if 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->NewDataAvailable == FALSE means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P05STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent a timeout.

7.6.7 E2E_P05Protect

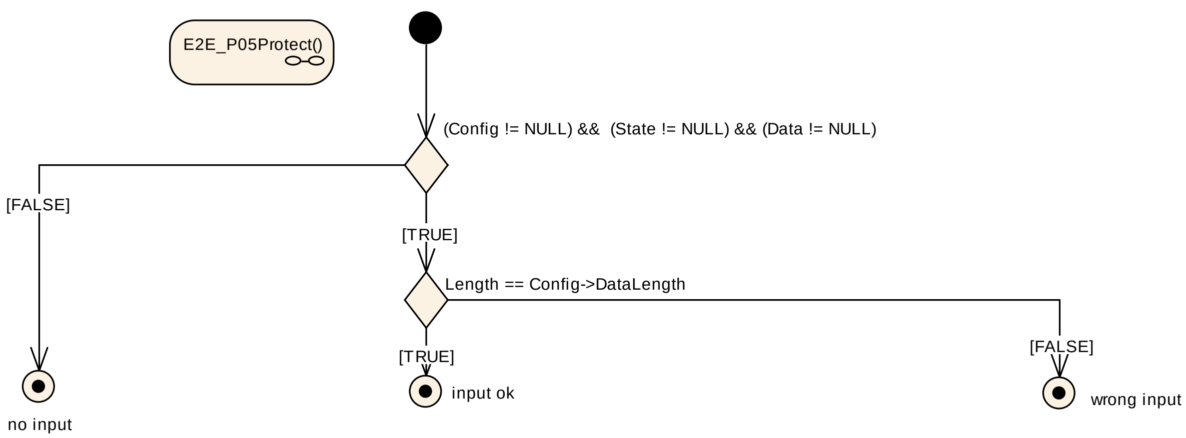
The function E2E_P05Protect() performs the steps as specified by the following six diagrams in this section.

[SWS_E2E_00403]⌈ The function E2E_P05Protect() shall have the following overall behavior:



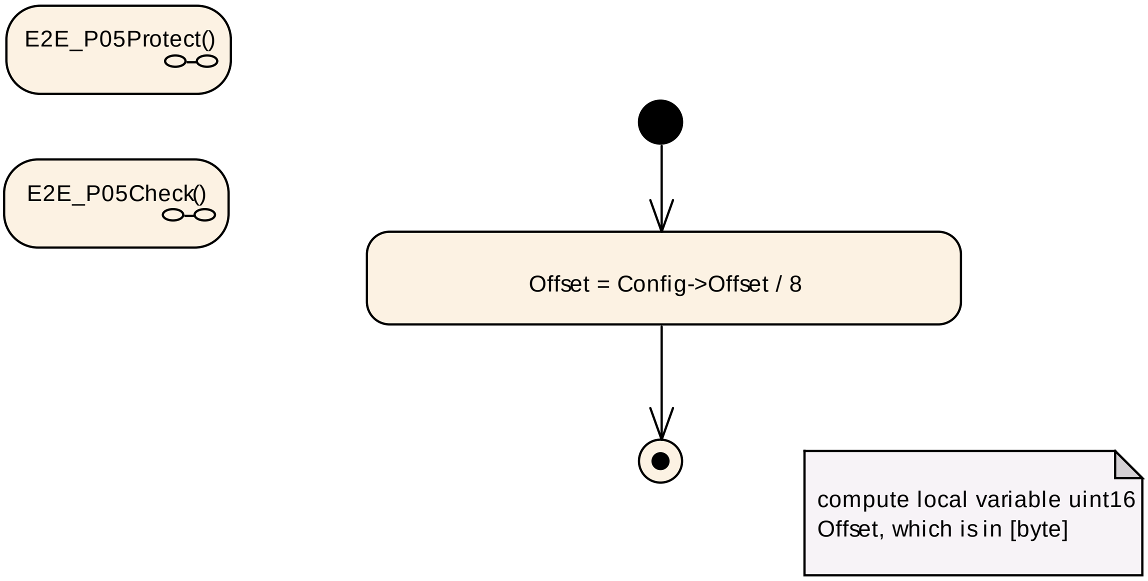
⌋ (SRS_E2E_08539)

[SWS_E2E_00404]⌈ The step "Verify inputs of the protect function" in E2E_P05Protect() shall have the following behavior:



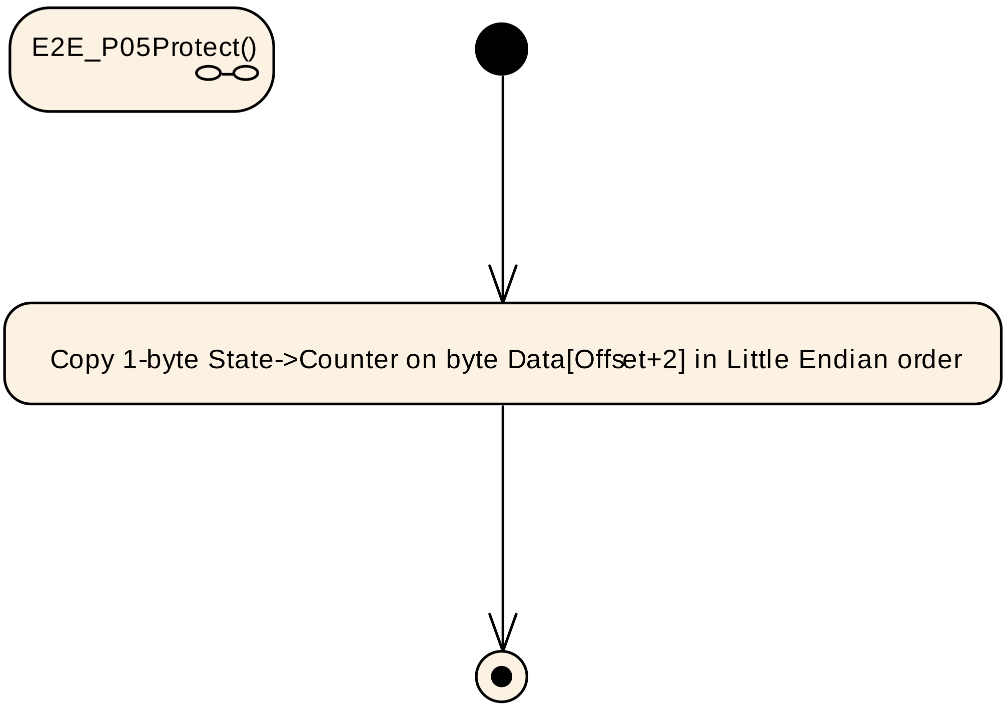
⌋ (SRS_E2E_08539)

[SWS_E2E_00469] ⌈ The step “Compute offset” in E2E_P05Protect() and E2E_P05Check() shall have the following behavior:



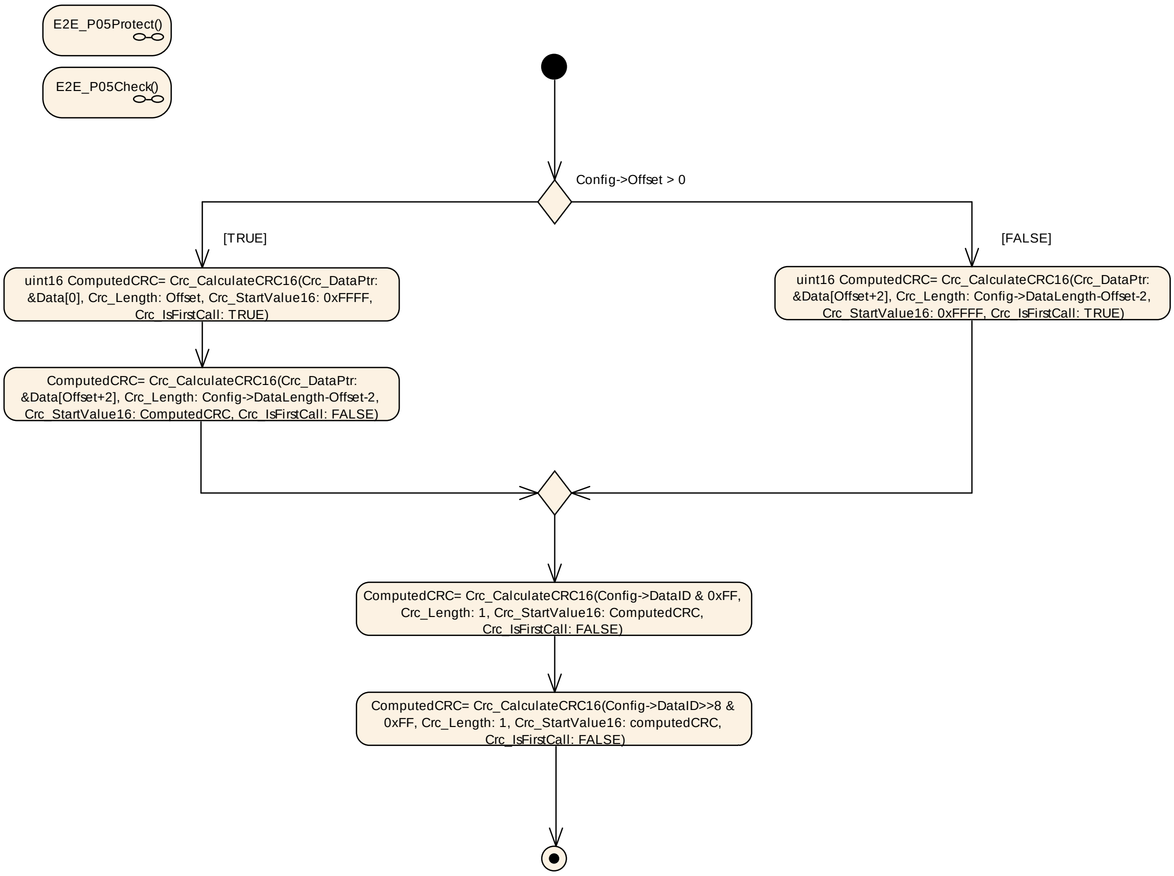
⌋ (SRS_E2E_08539)

[SWS_E2E_00405]⌈ The step “Write Counter” in E2E_P05Protect() shall have the following behavior:



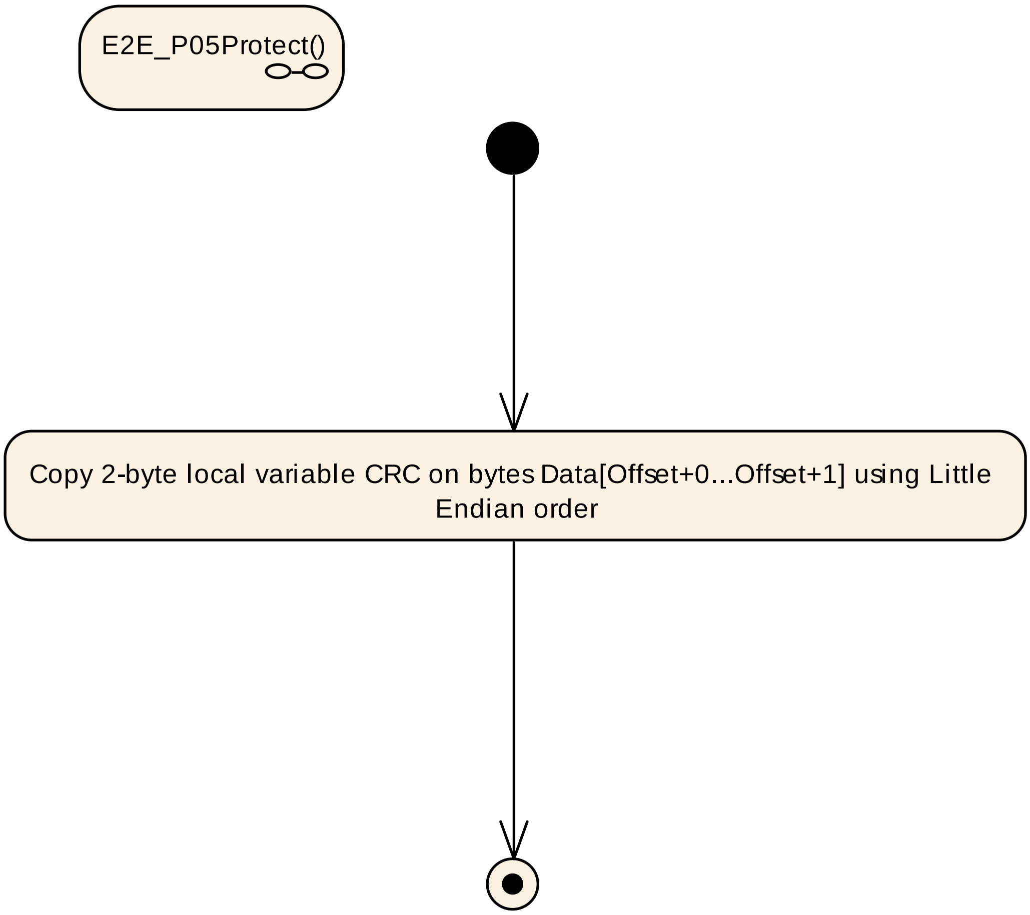
⌋ (SRS_E2E_08539)

[SWS_E2E_00406]⌈ The step “Compute CRC” in E2E_P05Protect() and in E2E_P05Check shall have the following behavior:



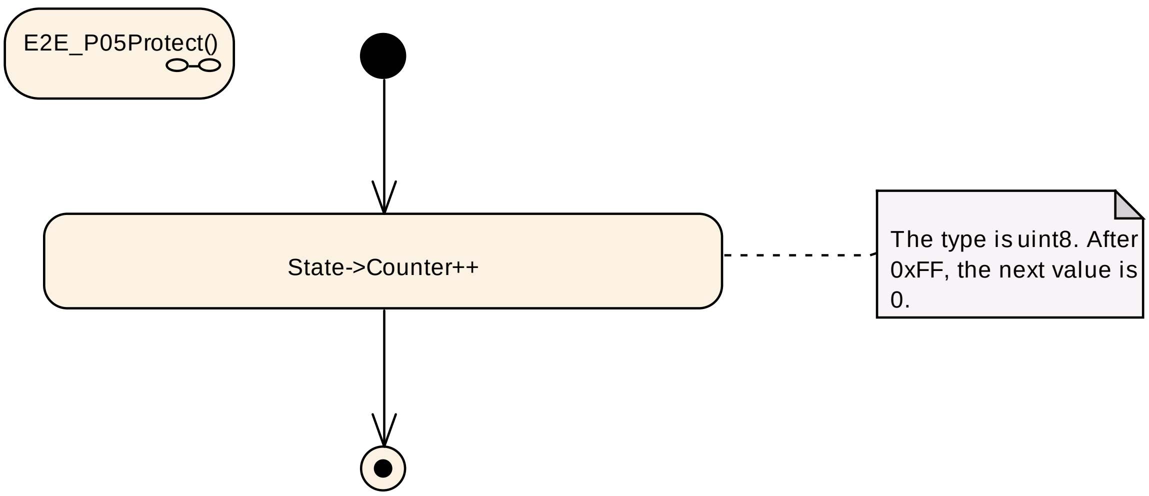
⌋ (SRS_E2E_08539)

[SWS_E2E_00407]⌈ The step “Write CRC” in E2E_P05Protect() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00409]⌈ The step “Increment Counter” in E2E_P05Protect() shall have the following behavior:

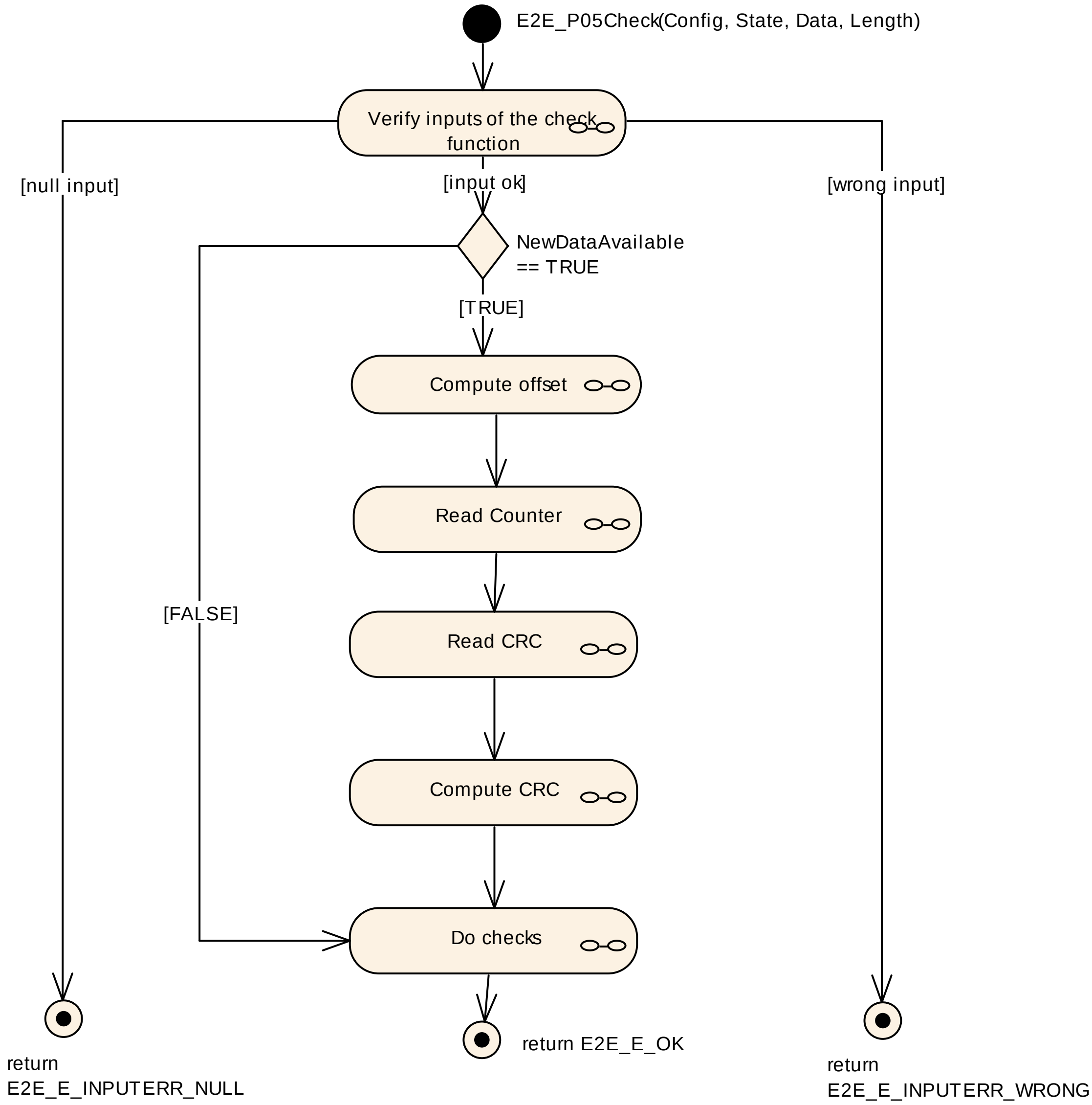


⌋ (SRS_E2E_08539)

7.6.8 E2E_P05Check

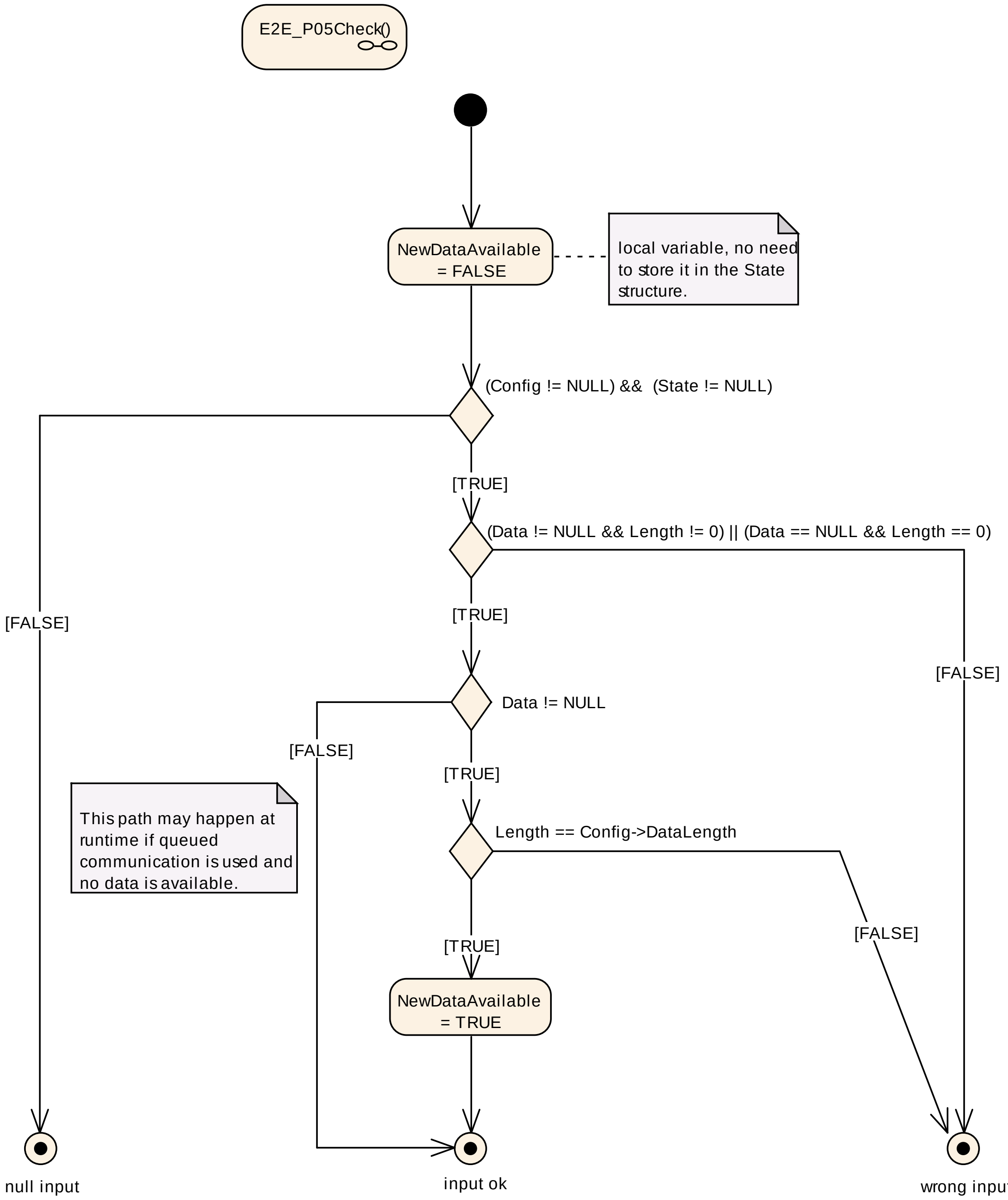
The function E2E_P05Check performs the actions as specified by the following six diagrams in this section.

[SWS_E2E_00411]⌈ The function E2E_P05Check() shall have the following overall behavior:



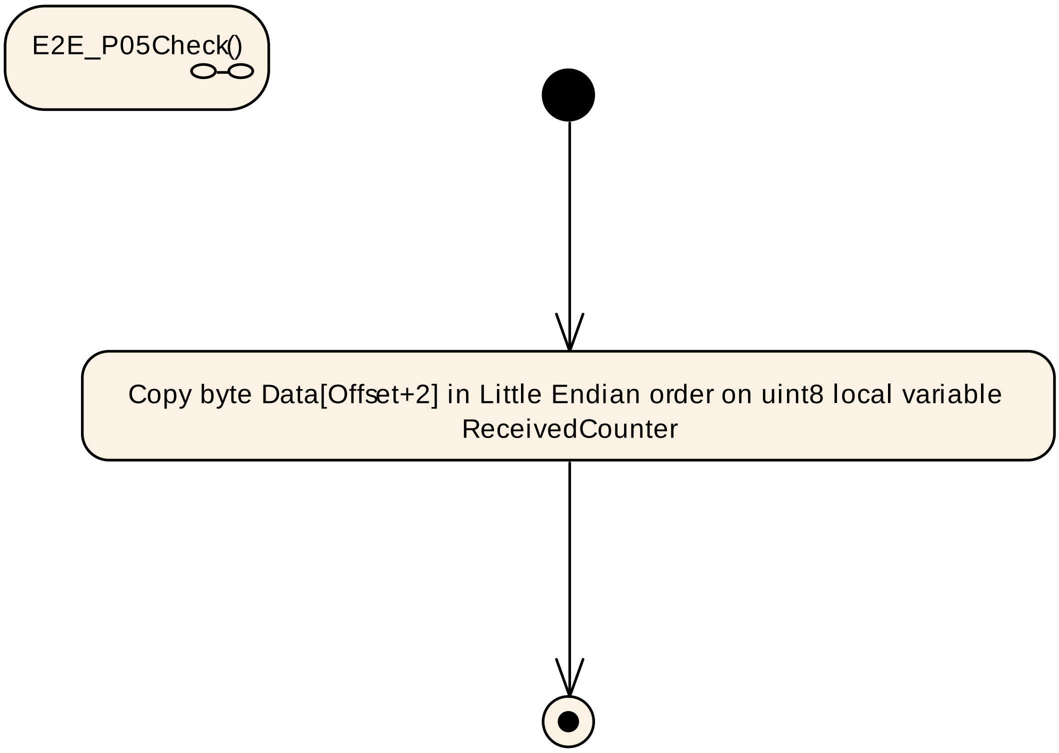
⌋ (SRS_E2E_08539)

[SWS_E2E_00412]⌈ The step "Verify inputs of the check function" in E2E_P05Check() shall have the following behavior:



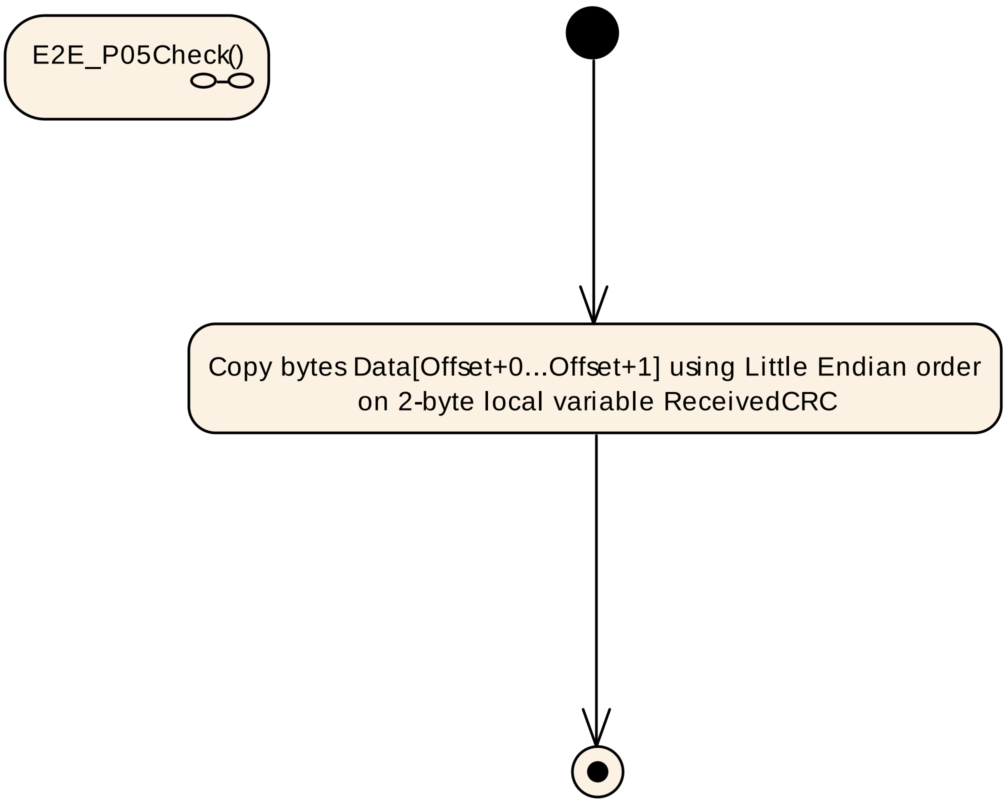
⌋ (SRS_E2E_08539)

[SWS_E2E_00413]⌈ The step “Read Counter” in E2E_P05Check() shall have the following behavior:



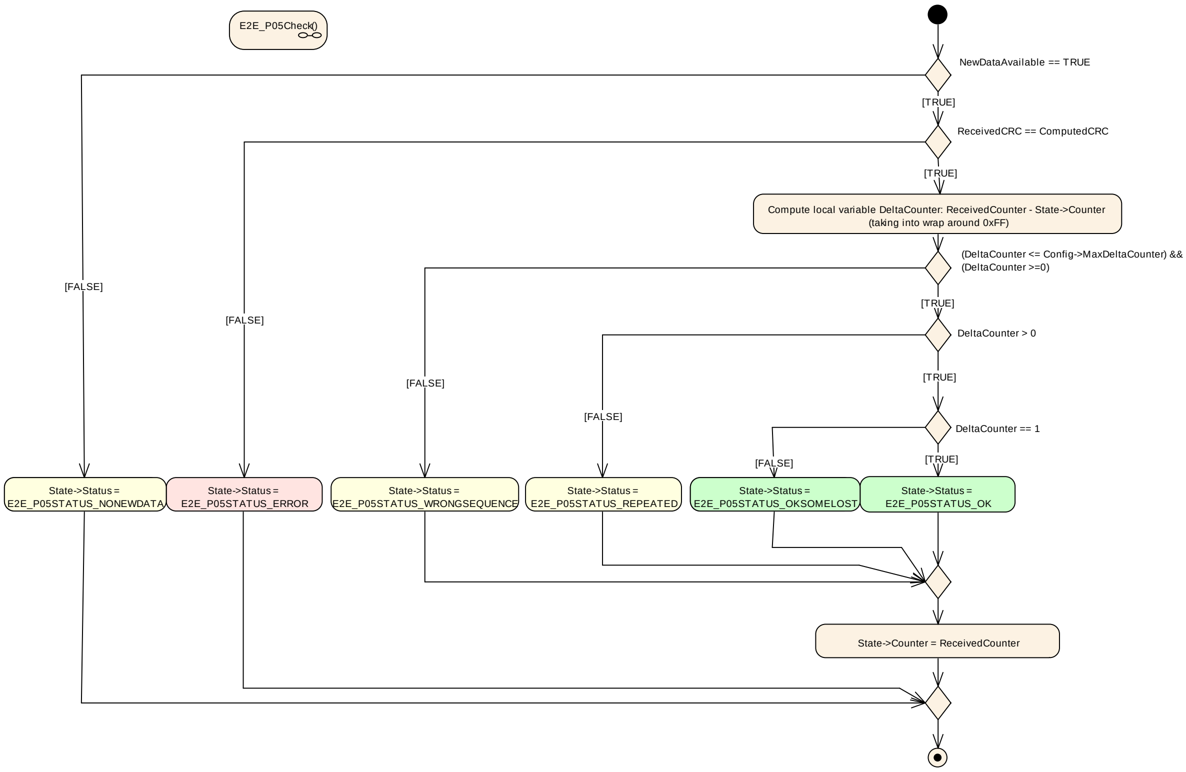
⌋ (SRS_E2E_08539)

[SWS_E2E_00414]⌈ The step "Read CRC" in E2E_P05Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00416]⌈ The step “Do Checks” in E2E_P05Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

7.7 Specification of E2E Profile 6

[SWS_E2E_00479]⌈ Profile 6 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Length	16 bits, to support dynamic-size data. (explicitly sent)
Counter	8-bits. (explicitly sent)
CRC	16-bits, polynomial in normal form 0x1021 (Autosar notation), provided by CRC library. (explicitly sent)
Data ID	16-bits, unique system-wide. (implicitly sent)

⌋ (SRS_E2E_08529, SRS_E2E_08533)

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

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC

Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 7-5: Detectable communication faults using Profile 6

For details of CRC computation, the usage of start values and XOR values see CRC Library [7].

7.7.1 Data Layout

7.7.1.1 User data layout

In the E2E Profile 6, the user data layout (of the data to be protected) is not constrained by E2E Profile 6 – there is only a requirement that the length of data to be protected is multiple of 1 byte.

7.7.1.2 Header layout

The header of the E2E Profile 6 has one fixed layout, as follows:

Transmission order	0								1								2								3							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	E2E CRC																E2E Length															
4	E2E Counter																															

Figure 7-15: E2E Profile 6 header

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte fist), applicable for both implicit and explicit header fields – imposed by profile
2. LSB Fist (least significant bit within byte first) – imposed by TCP/IP bus

7.7.2 Counter

In E2E Profile 6, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E library.

[SWS_E2E_00417]⌈ In E2E Profile 6, 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. When the counter reaches the maximum value (0xFF), then it shall restart with 0 for the next send request.⌋ (SRS_E2E_08539)

Note that the counter value 0xFF is not reserved as a special invalid value, but it is used as a normal counter value.

[SWS_E2E_00473]⌈ In E2E Profile 6, on the receiver side, the counter shall be initialized with 0xFF.⌋ (SRS_E2E_08539)

In E2E Profile 6, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:
    a. no new data has arrived since last invocation of E2E library check function,
    b. the data is repeated
2. OK:
    a. counter is incremented by one (i.e. no data lost),
    b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),
3. Error:
    a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

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

7.7.3 Data ID

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

[SWS_E2E_00419]⌈ In the E2E Profile 5, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation.⌋ (SRS_E2E_08539)

The Data ID is not a part of the transmitted E2E header (similar to Profile 2 and 5).

[UC_E2E_00464]⌈ In the E2E profile 6, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data).
⌋ (SRS_E2E_08539)

In case of usage of E2E Library for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Library using Data ID.

In case of usage of E2E Library for protecting I-PDUs (i.e. invocation from COM), the receiver COM expects at a reception only a specific I-PDU, which is checked by E2E Library using Data ID.

7.7.4 Length

In Profile 6 the length field is introduced to support variable-size length – the Data[] array storing the serialized data can potentially have a different length in each cycle.
In Profile 6 there is a explicit transmission of the length.

7.7.5 CRC

E2E Profile 6 uses a 16-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[SWS_E2E_00420]⌈ E2E Profile 6 shall use the Crc_CalculateCRC16() function of the SWS CRC Library for calculating the CRC (Polynomial: 0x1021; Autosar notation).⌋ (SRS_E2E_08539, SRS_E2E_08531)

[SWS_E2E_00421]⌈ In E2E Profile 6, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended with the Data ID.
⌋ (SRS_E2E_08539, SRS_E2E_08536)

7.7.6 Timeout detection

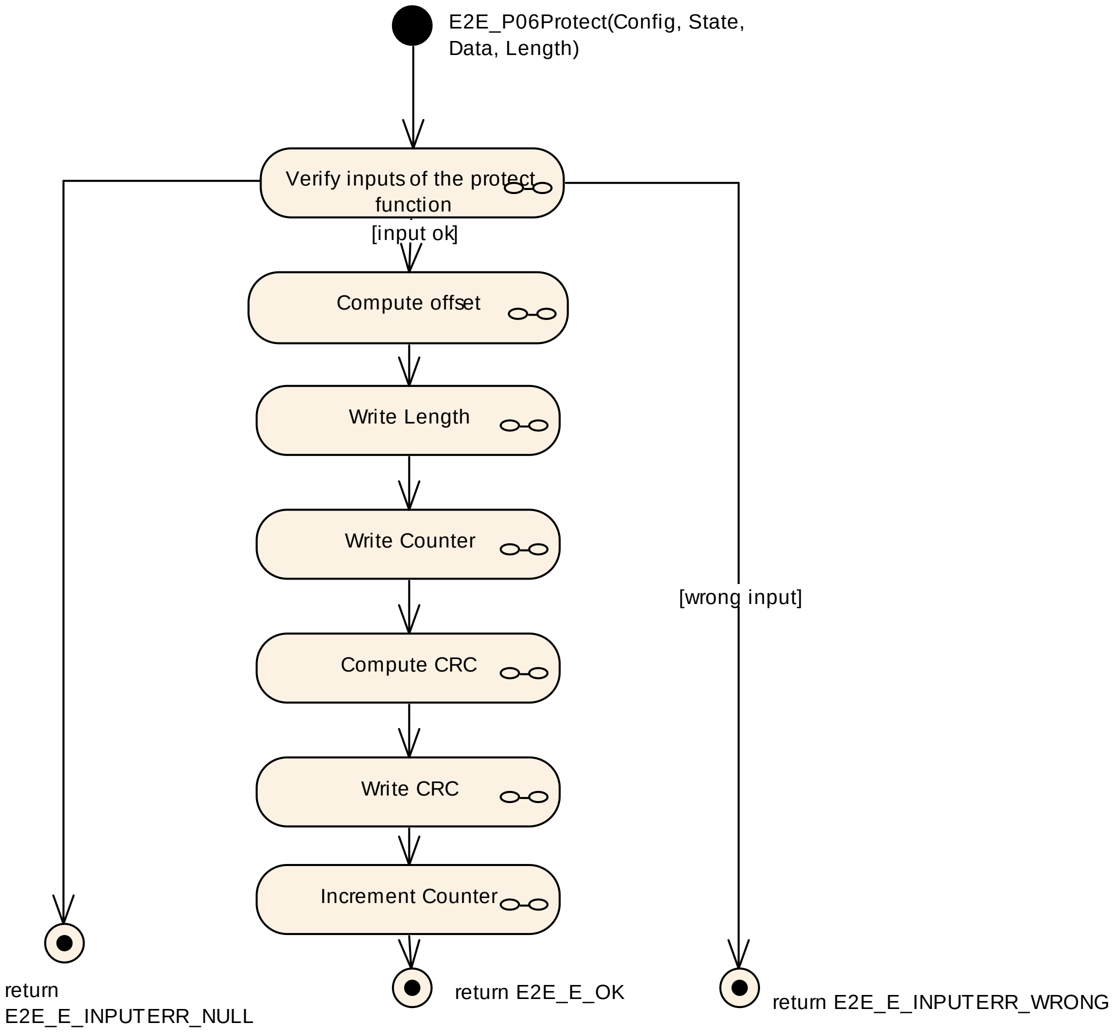
The previously mentioned mechanisms (for Profile 6: CRC, Counter, Data ID, Length) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead 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->NewDataAvailable == FALSE means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P06STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent a timeout.

7.7.7 E2E_P06Protect

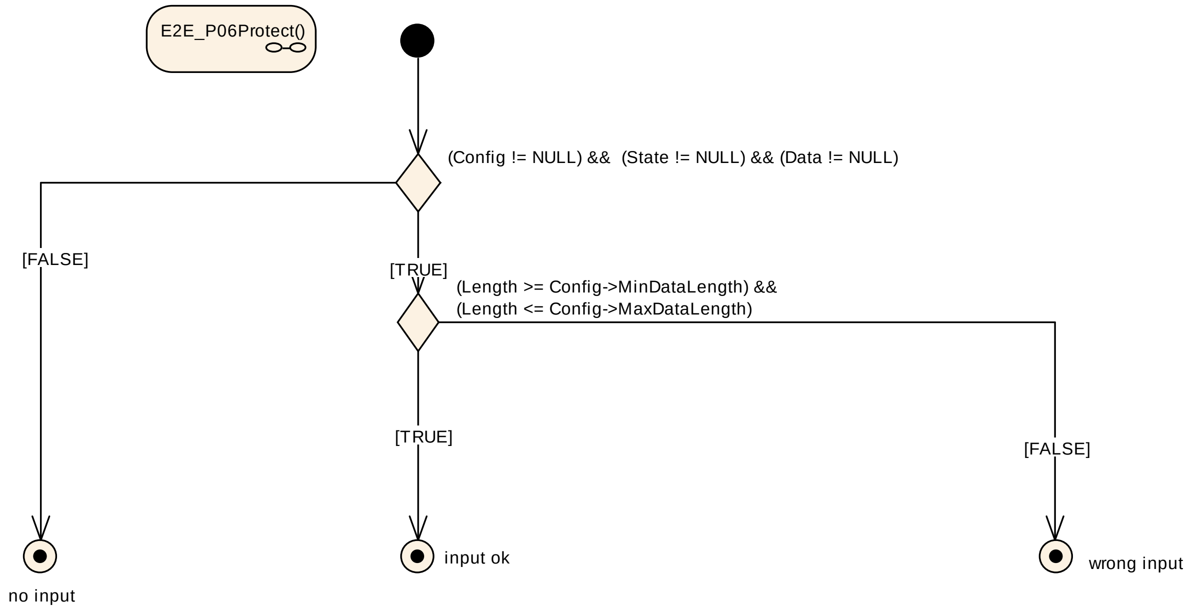
The function E2E_P06Protect() performs the steps as specified by the following seven diagrams in this section.

[SWS_E2E_00423]⌈ The function E2E_P06Protect() shall have the following overall behavior:



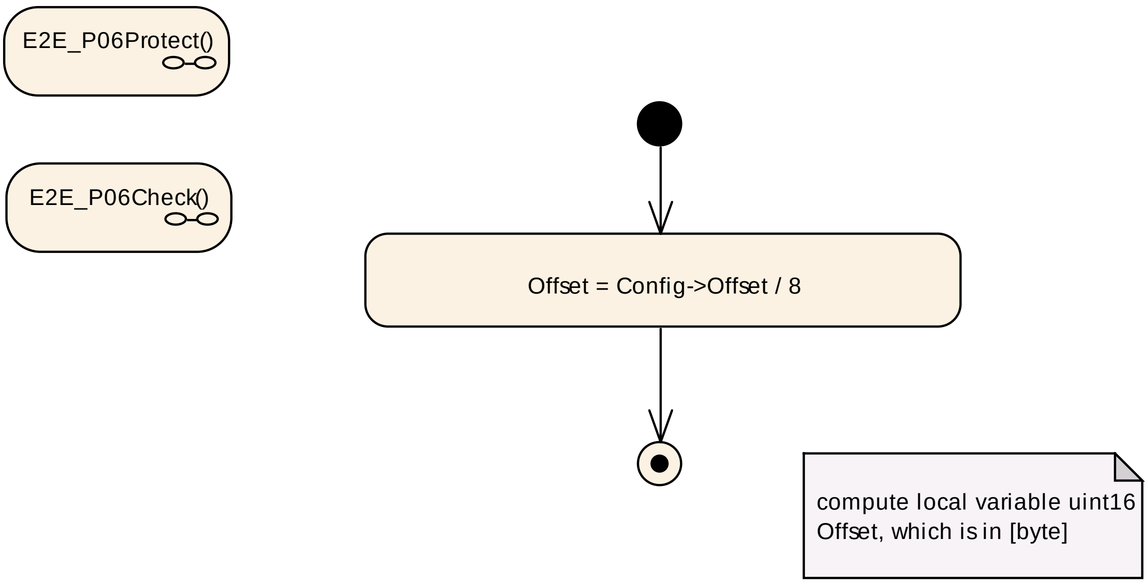
⌋ (SRS_E2E_08539)

[SWS_E2E_00424]⌈ The step “Verify inputs of the protect function” in E2E_P06Protect() shall have the following behavior:



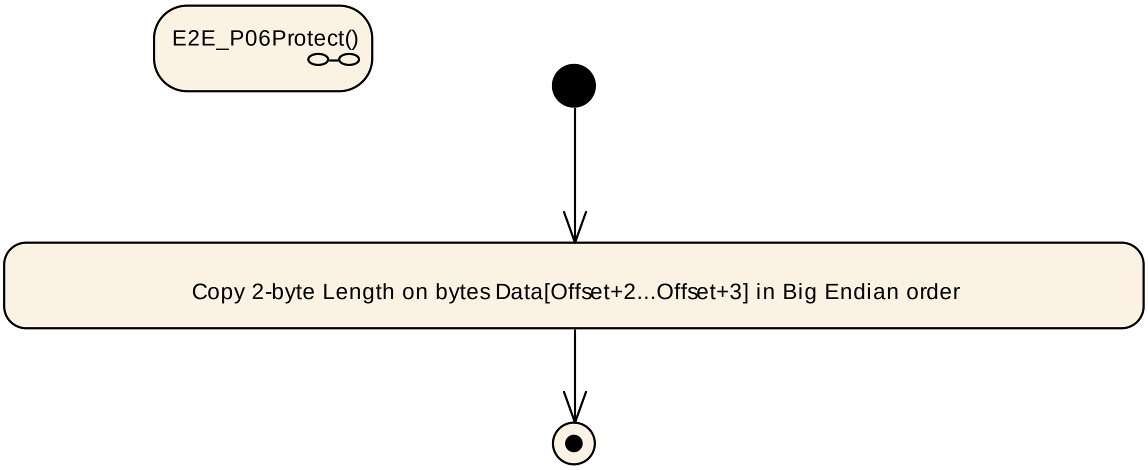
⌋ (SRS_E2E_08539)

[SWS_E2E_00470]⌈ The step “Compute offset” in E2E_P06Protect() and E2E_P06Check() shall have the following behavior:



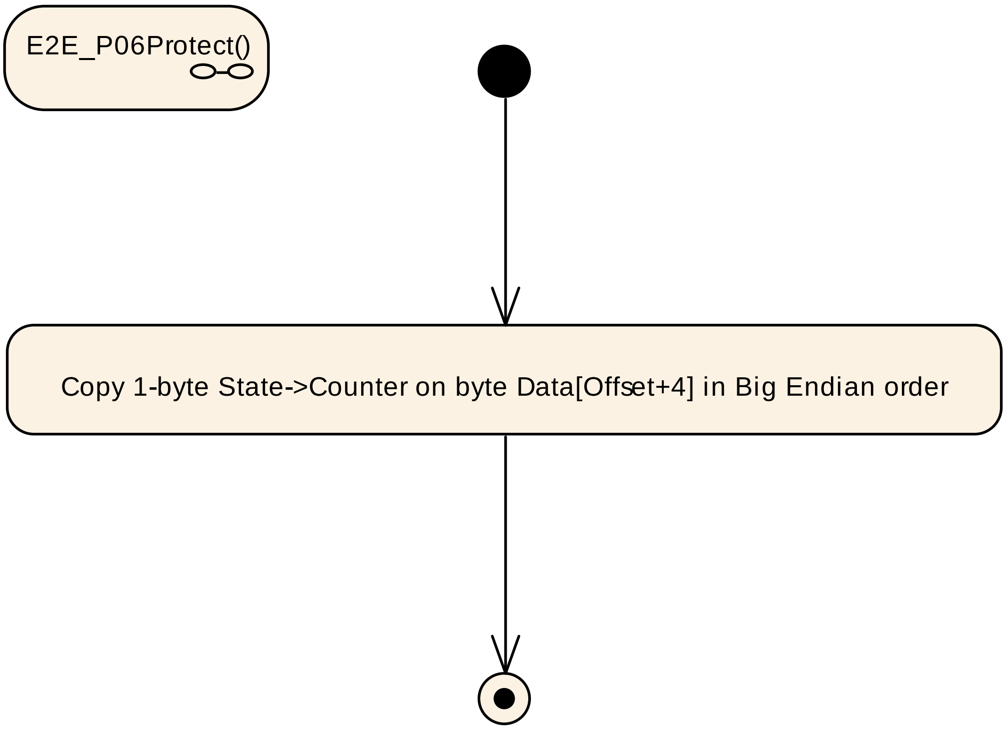
⌋ (SRS_E2E_08539)

[SWS_E2E_00425]⌈ The step “Write Length” in E2E_P06Protect() shall have the following behavior:



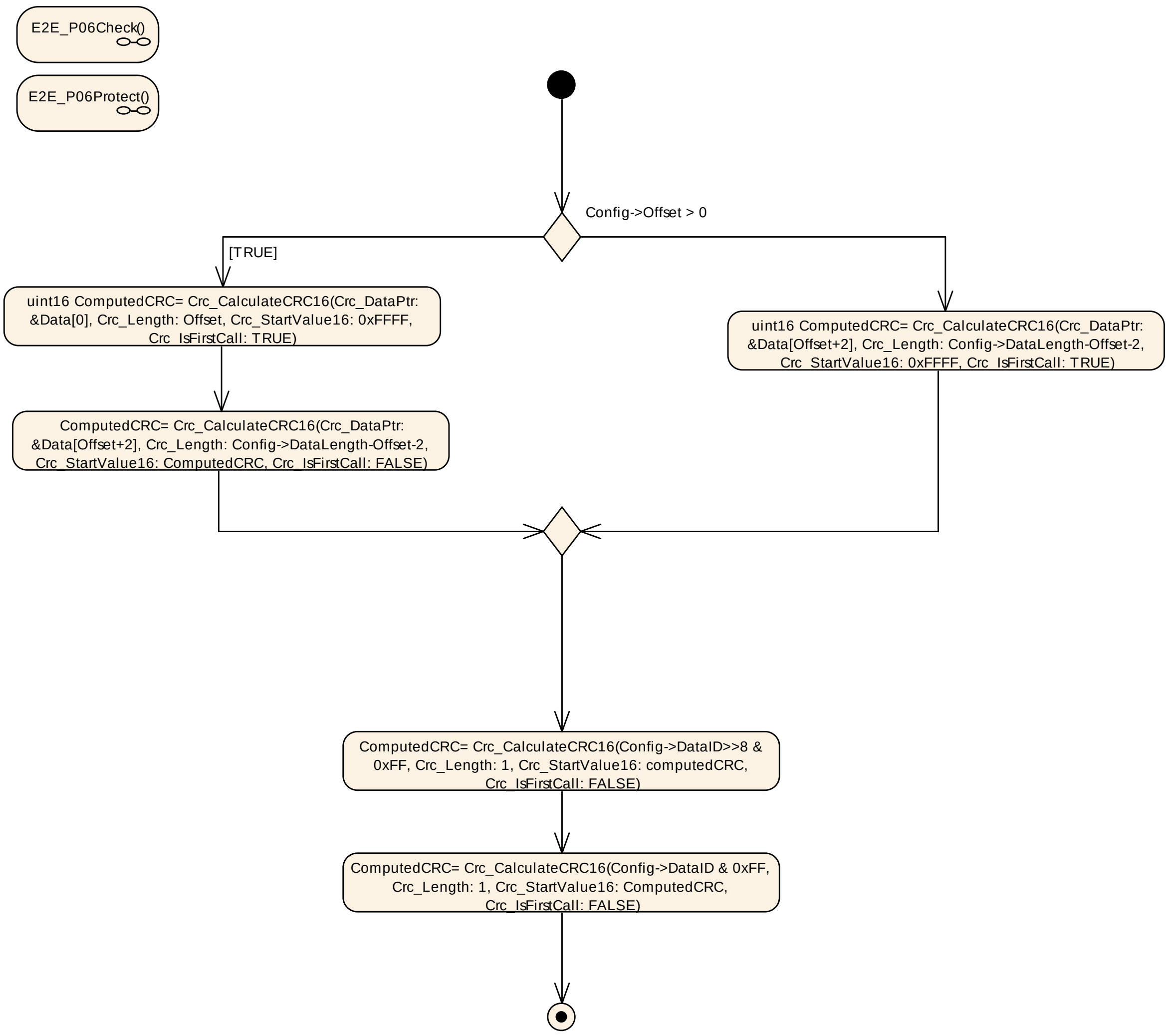
⌋ (SRS_E2E_08539)

[SWS_E2E_00426]⌈ The step “Write Counter” in E2E_P06Protect() shall have the following behavior:



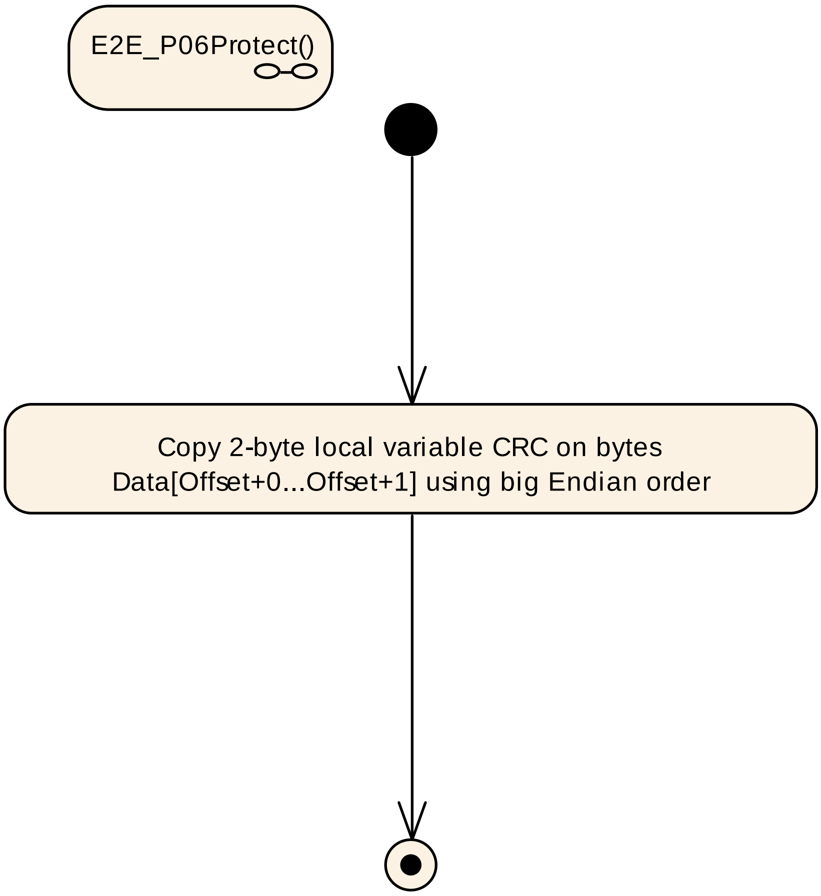
⌋ (SRS_E2E_08539)

[SWS_E2E_00427]⌈ The step “Compute CRC” in E2E_P06Protect() and E2E_P06Check() shall have the following behavior:



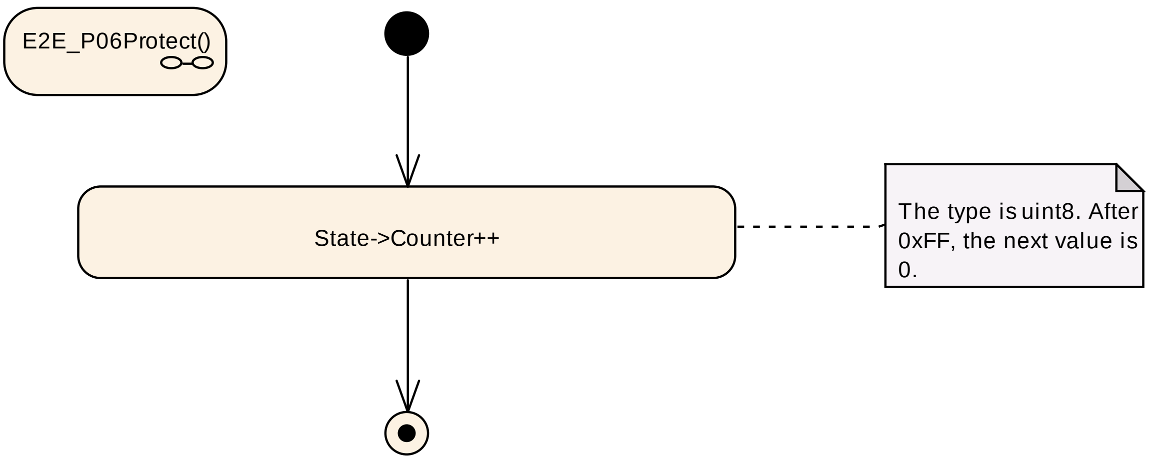
⌋ (SRS_E2E_08539)

[SWS_E2E_00428]⌈ The step "Write CRC" in E2E_P06Protect() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00429]⌈ The step “Increment Counter” in E2E_P06Protect() shall have the following behavior:

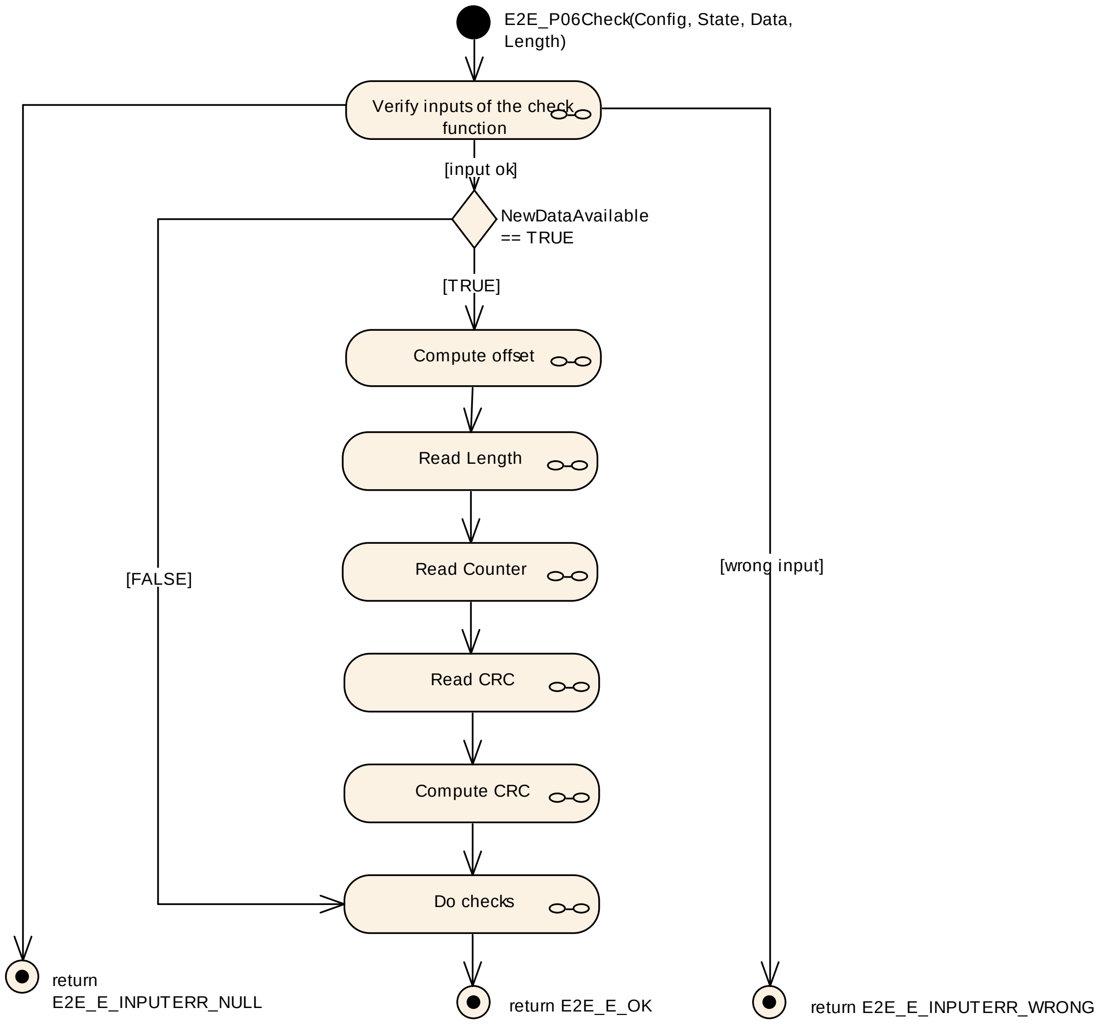


⌋ (SRS_E2E_08539)

7.7.8 E2E_P06Check

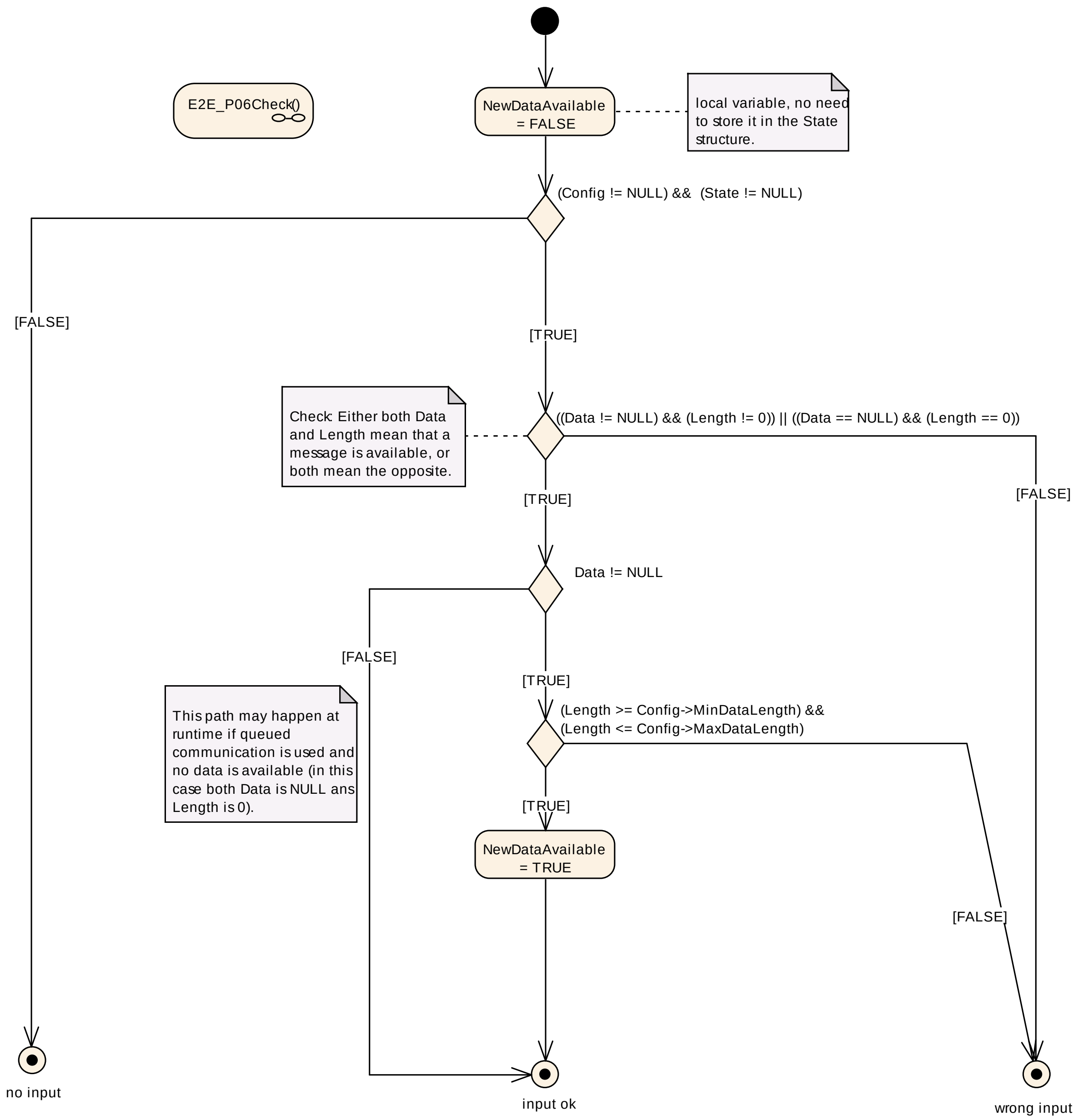
The function E2E_P06Check performs the actions as specified by the following seven diagrams in this section.

[SWS_E2E_00430]⌈ The function E2E_P06Check() shall have the following overall behavior:



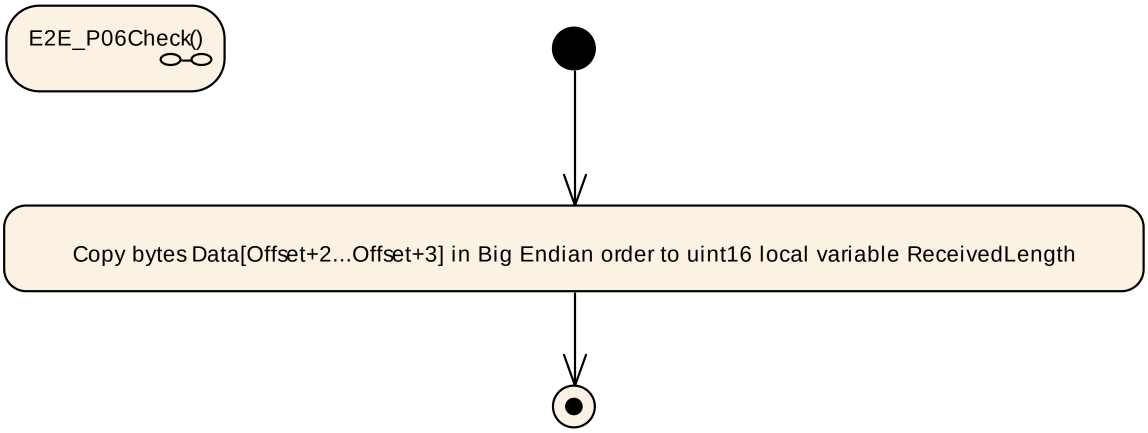
⌋ (SRS_E2E_08539)

[SWS_E2E_00431]⌈ The step "Verify inputs of the check function" in E2E_P06Check() shall have the following behavior:



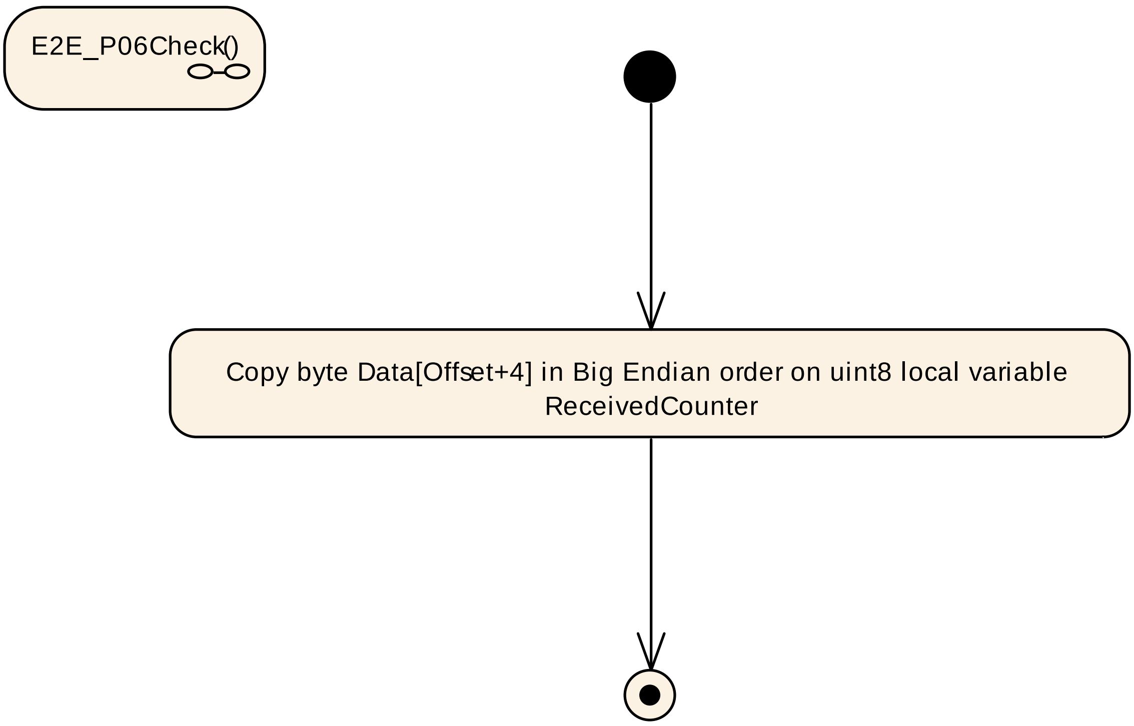
⌋ (SRS_E2E_08539)

[SWS_E2E_00432]⌈ The step “Read Length” in E2E_P06Check() shall have the following behavior:



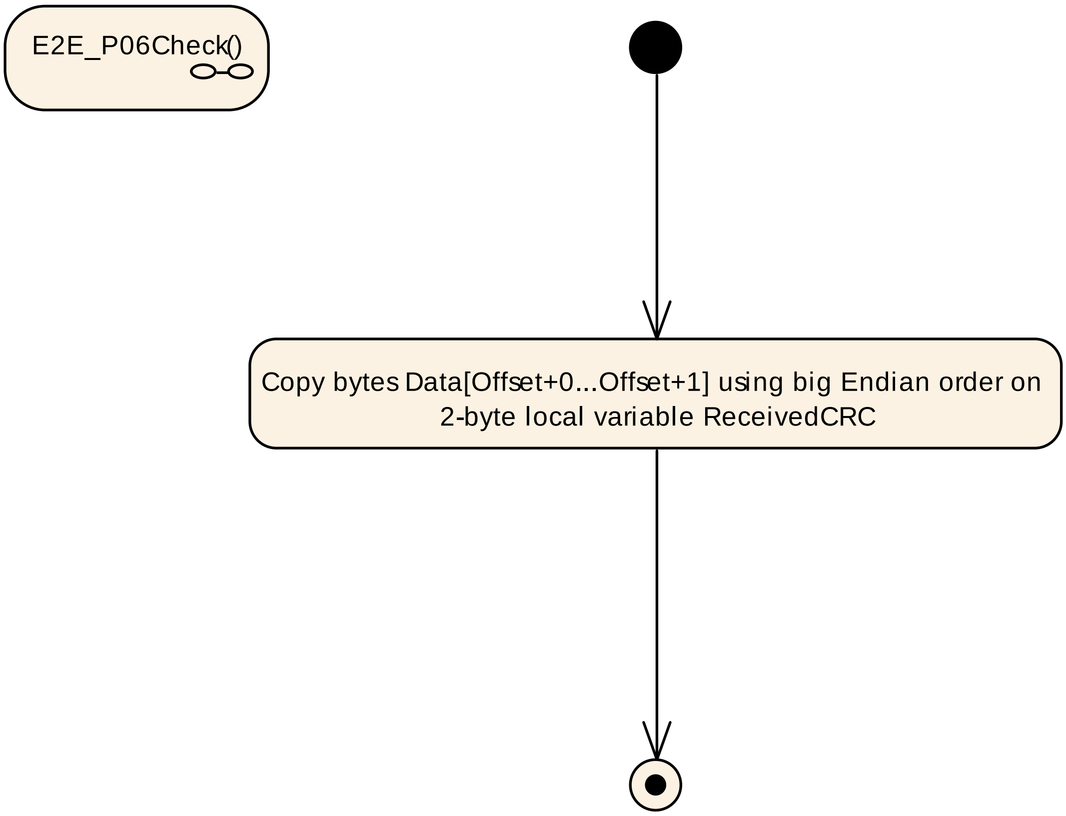
⌋ (SRS_E2E_08539)

[SWS_E2E_00433]⌈ The step “Read Counter” in E2E_P06Check() shall have the following behavior:



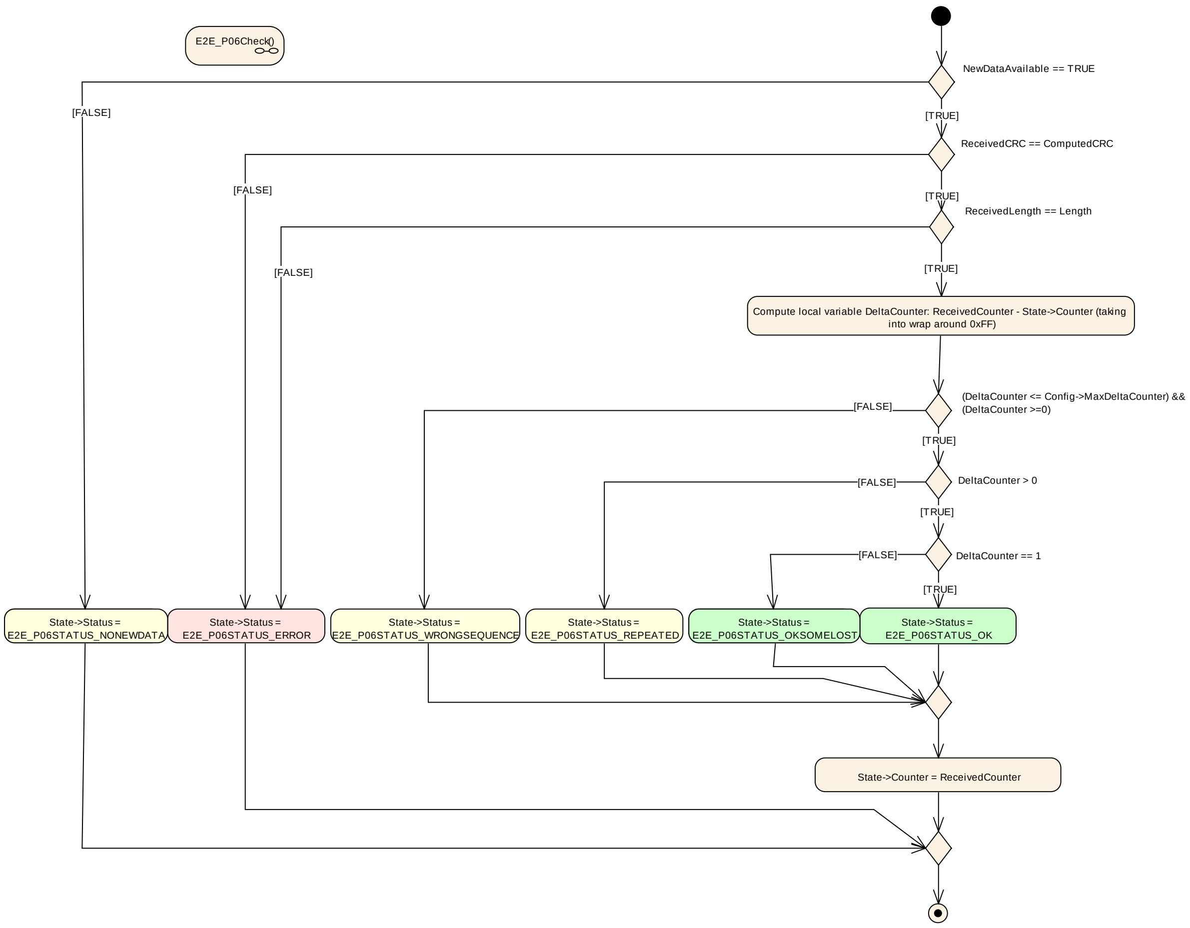
⌋ (SRS_E2E_08539)

[SWS_E2E_00434]⌈ The step “Read CRC” in E2E_P06Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

[SWS_E2E_00436]⌈ The step "Do Checks" in E2E_P06Check() shall have the following behavior:



⌋ (SRS_E2E_08539)

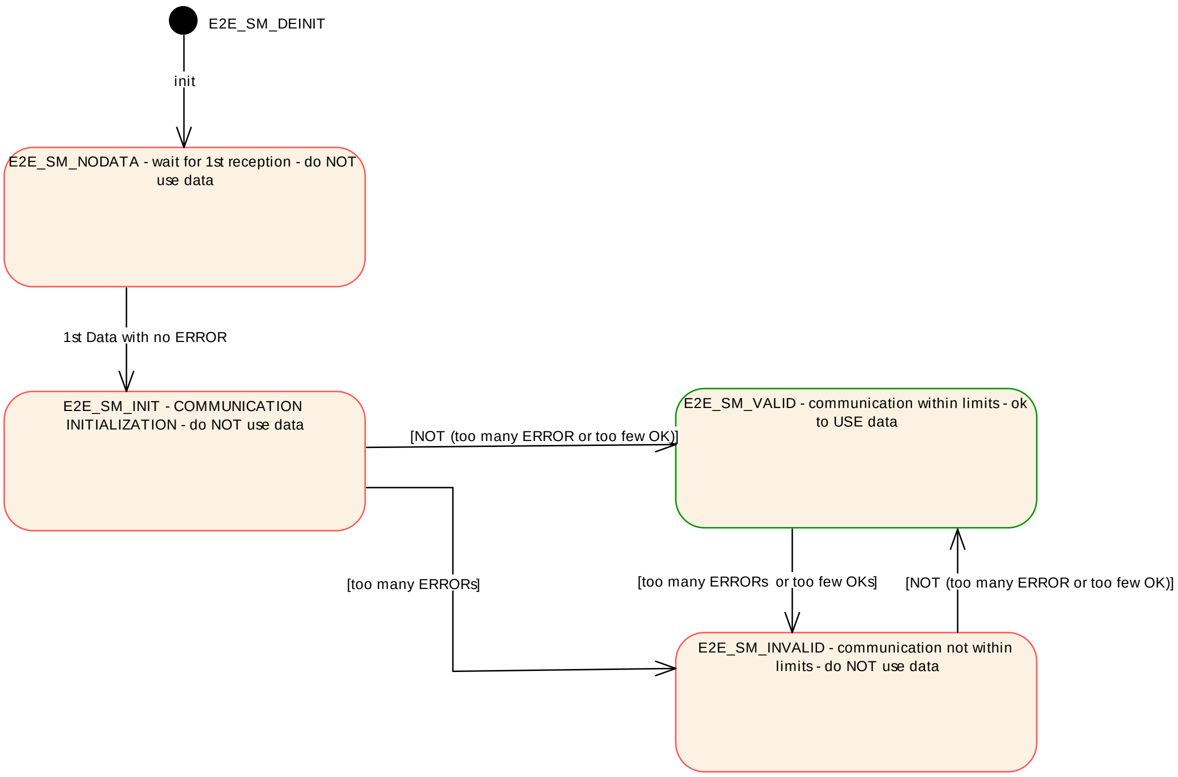
7.8 Specification of E2E state machine

The E2E Profile check()-functions verifies data in one cycle. This function only determines if data in that cycle are correct or not. In contrary, the state machine builds up a state out of several results of check() function within a reception window, which is then provided to the consumer (RTE/SWC/COM).

The state machine is applicable for all E2E profiles. Profiles P1 and P2 can be configured to work together with the state machine. However, the behavior of P1/P2 alone, regardless how it is configured, is different to the behavior of P1/P2 + state machine.

7.8.1 Overview of the state machine

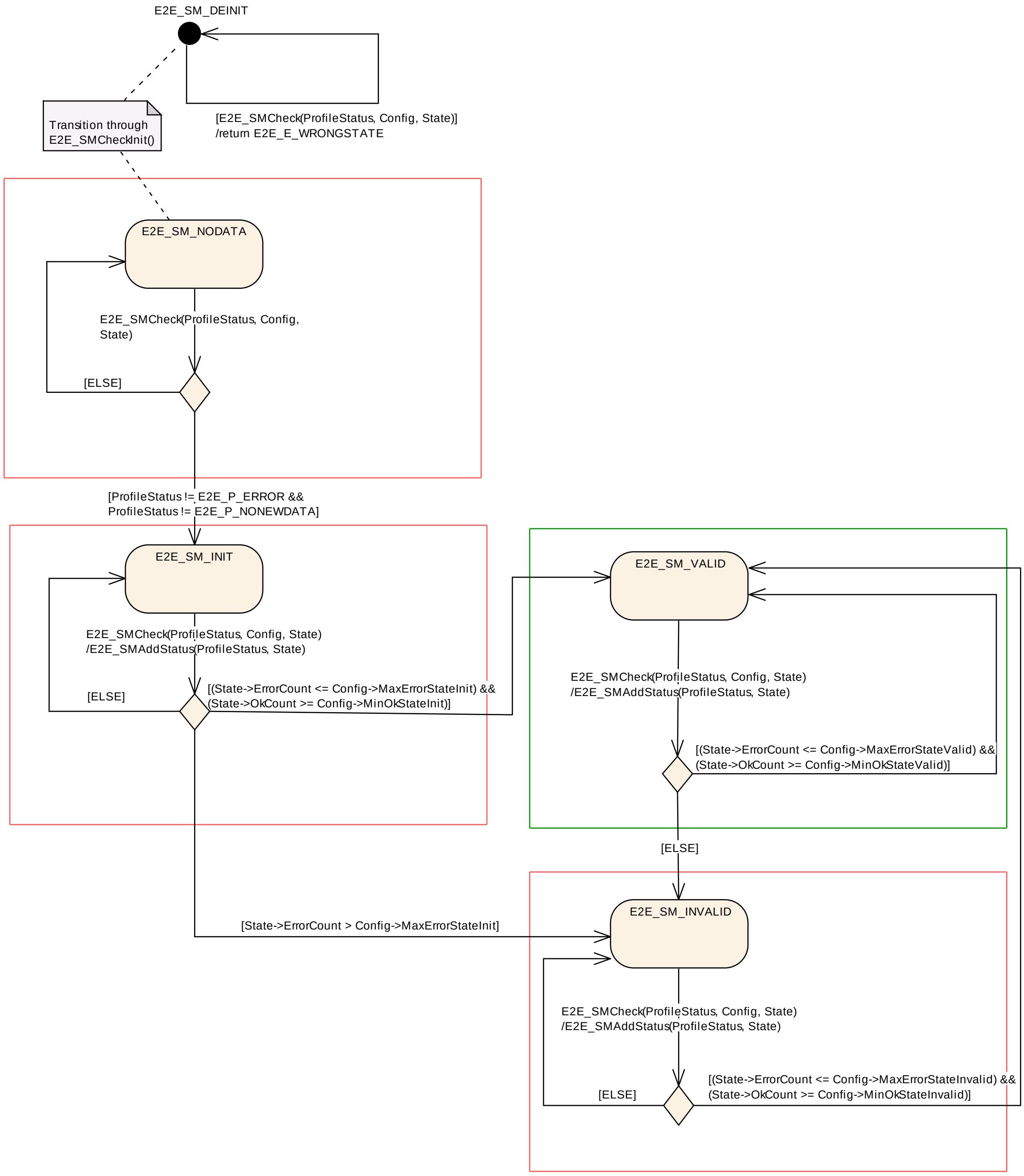
The diagram below summarizes the state machine.



7.8.2 State machine specification

[SWS_E2E_00354]⌈ The E2E state machine shall be implemented by the functions E2E_SMCheck() and E2E_SMCheckInit().⌋ (SRS_E2E_08539)

[SWS_E2E_00345]⌈ The E2E State machine shall have the following behavior with respect to the function E2E_SMCheck():

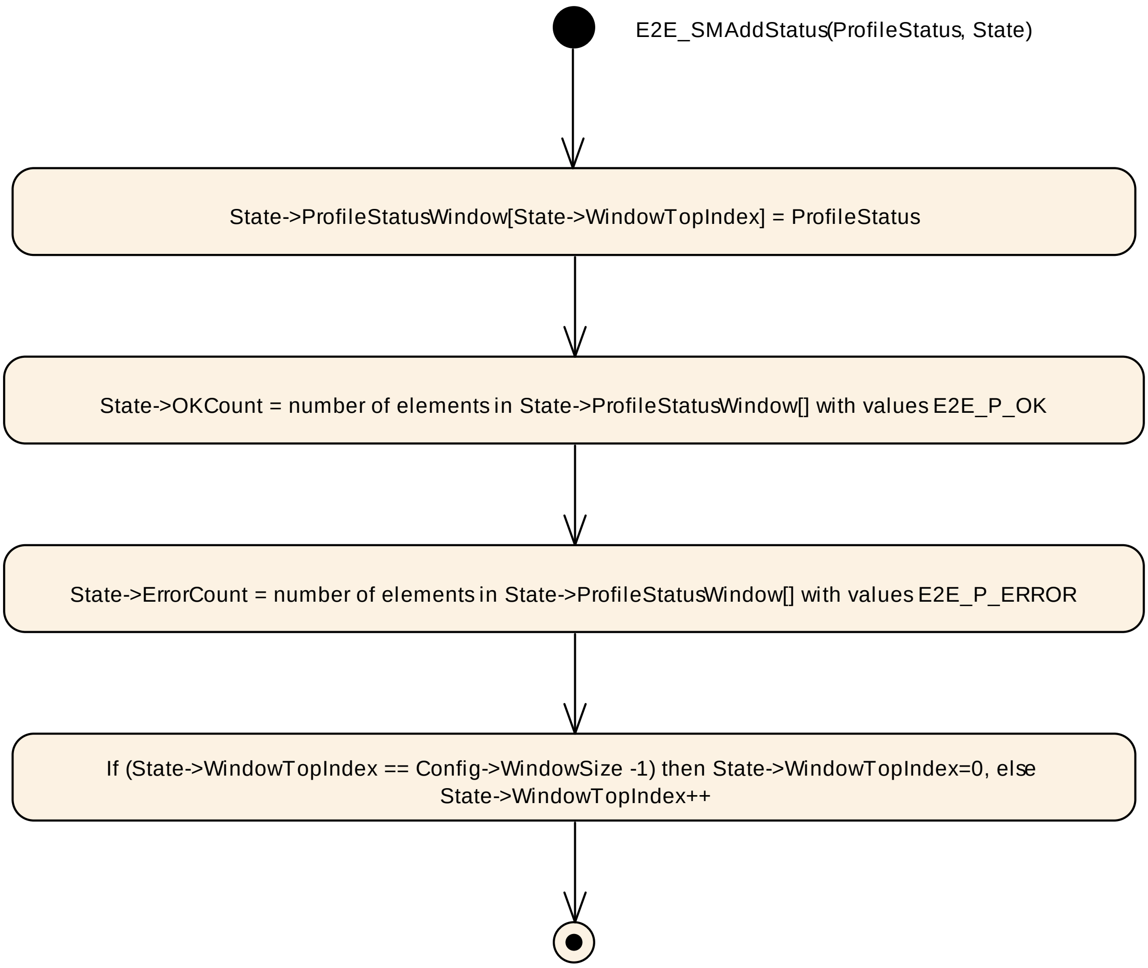


This shall be understood as follows:

1. The current state (e.g. E2E_SM_VALID) is stored in State->SMState
2. At every invocation of E2E_SMCheck, the ProfileStatus is processed (as shown by logical step E2E_SMAddStatus()
3. After that, there is an examination of two counters: State->ErrorCount and State->OKCount. Depending on their values, there is a transition to a new state, stored in State->SMState.

⌋ (SRS_E2E_08539)

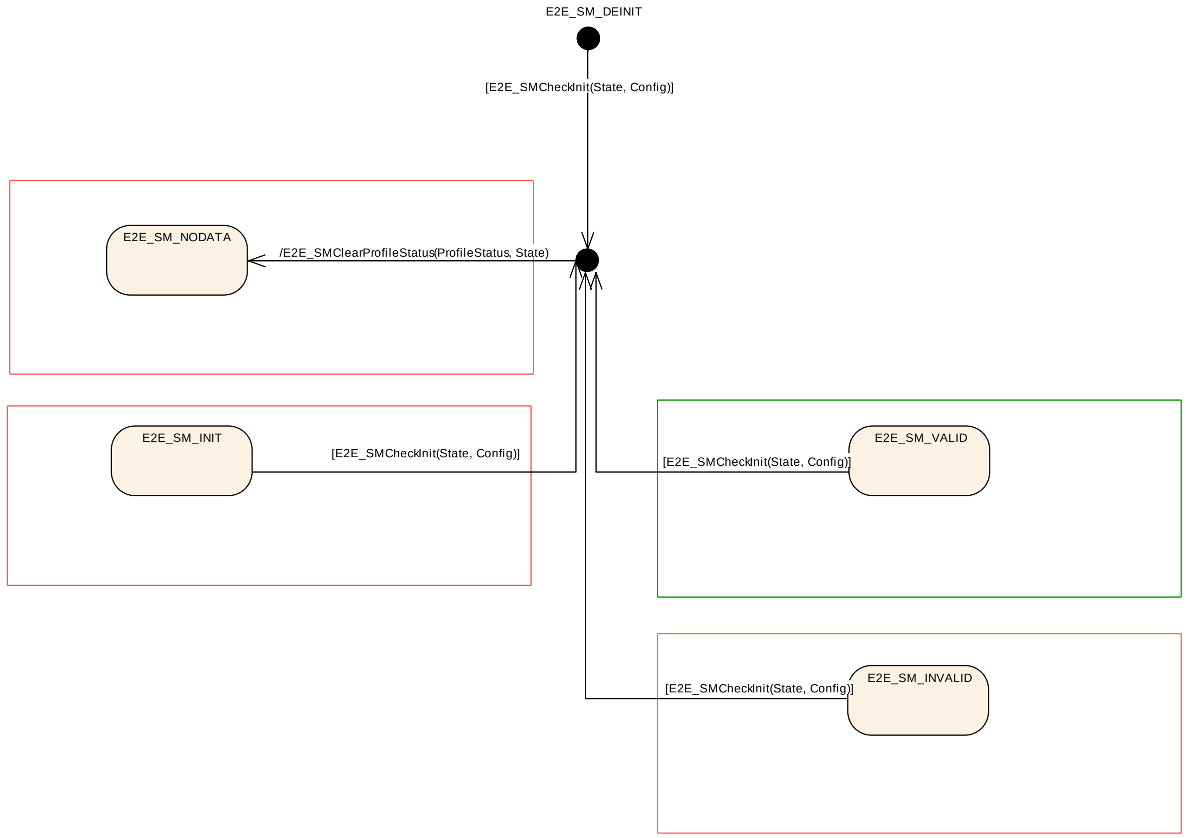
[SWS_E2E_00466]⌈ The step E2E_SMAddStatus(ProfileStatus, State) in E2E_SMCheck() shall have the following behavior:



⌋ (SRS_E2E_08539)

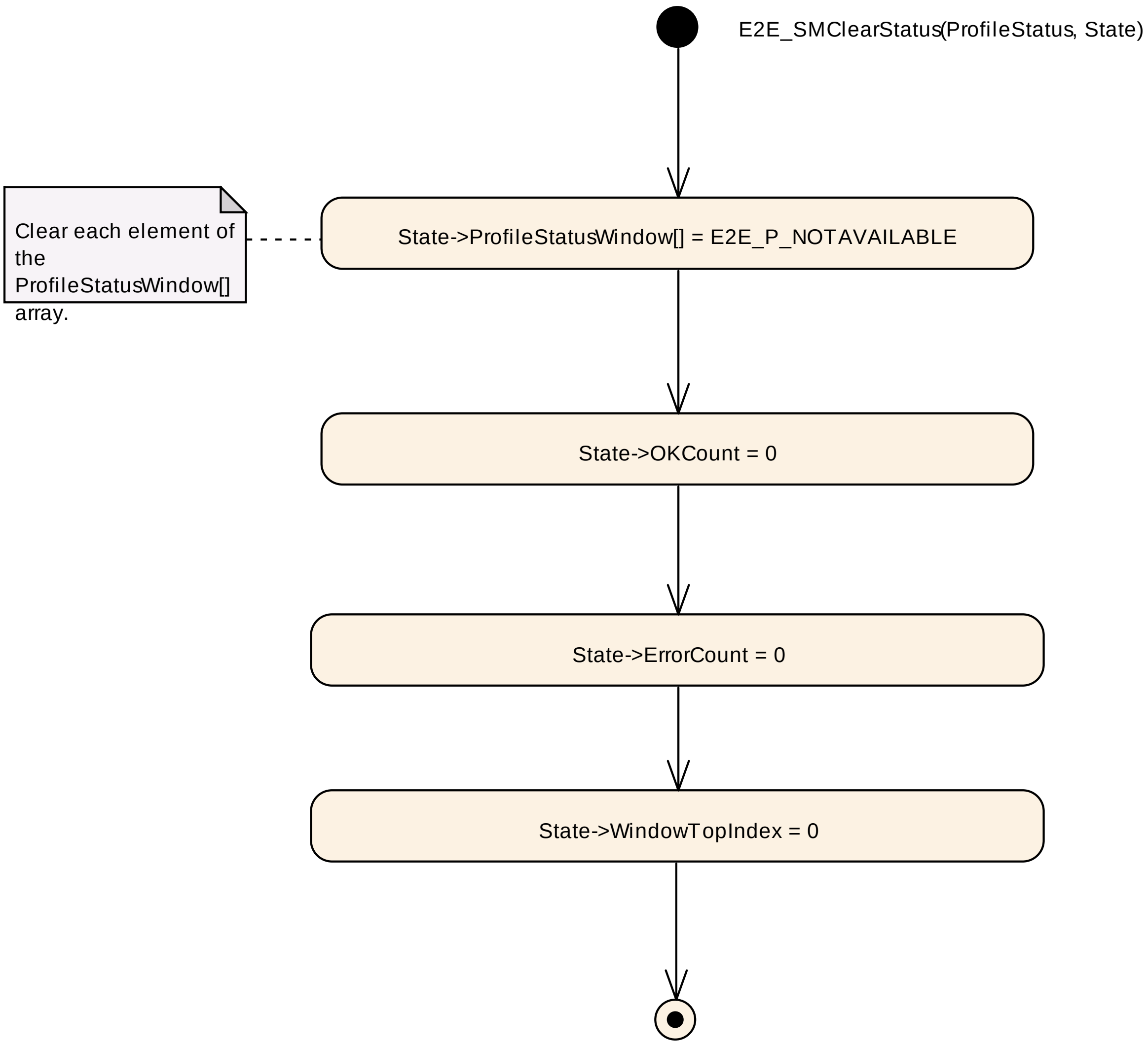
E2E_SMAddStatus is just a logical step in the algorithm, it may (but it does not have to be) implemented a a separate function. It is not a module API function.

[SWS_E2E_00375]⌈ The E2E State machine shall have the following behavior with respect to the function E2E_SMCheckInit():



⌋ (SRS_E2E_08539)

[SWS_E2E_00467]⌈ The step E2E_SMClearStatus(ProfileStatus, State) in E2E_SMCheck() shall have the following behavior:



⌋ (SRS_BSW_00003)

7.9 Version Check

[SWS_E2E_00327]⌈ The implementer of the E2E Library shall avoid the integration of incompatible files. Minimum implementation is the version check of the header files.
For included header files:

- E2E_AR_RELEASE_MAJOR_VERSION
- E2E_AR_RELEASE_MINOR_VERSION

shall be identical. For the module internal c and h files:

- E2E_SW_MAJOR_VERSION
- E2E_SW_MINOR_VERSION
- E2E_AR_RELEASE_MAJOR_VERSION
- E2E_AR_RELEASE_MINOR_VERSION
- E2E_AR_RELEASE_REVISION_VERSION

shall be identical (see also [SWS_E2E_00038] for published information). ⌋ (SRS_BSW_00003)

8 API specification

This chapter specifies the API of E2E Library.

Members of the configuration structures (e.g. in Figure 9-1) are in alphabetical order. However, for implementation, the sequence of members of this data structure is provided by table specification items (e.g. [SWS_E2E_00018]).

8.1 Imported types

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

[SWS_E2E_00017]⌈

Module	***Imported Type***
GENERIC TYPES	<InType>
Rte	Rte_Instance
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋ (SRS_E2E_08528)

8.2 Type definitions

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

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

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

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

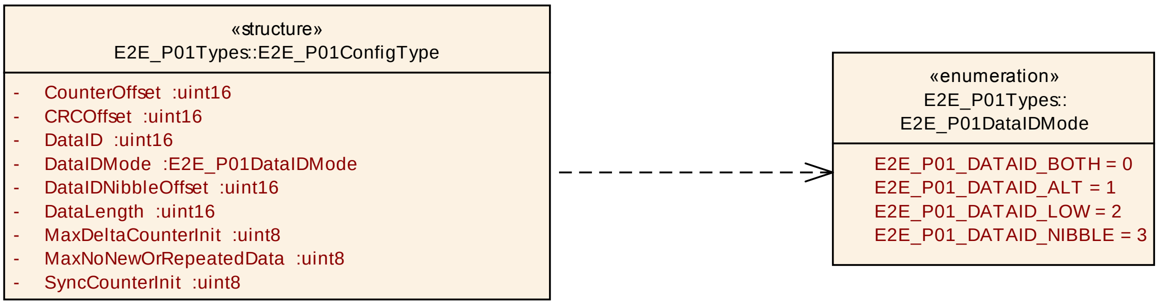


Figure 8-1: E2E Profile 1 configuration

8.2.1.1 E2E_P01ConfigType

[SWS_E2E_00018]⌈

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.		

⌋ (SRS_E2E_08528)

8.2.1.2 E2E_P01DataIDMode

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

[SWS_E2E_00200]⌈

Name:	E2E_P01DataIDMode	
Type:	Enumeration	
Range:	E2E_P01_DATAID_BOTH	Two bytes are included in the CRC (double ID configuration) This is used in E2E 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 E2E 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 four inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.

⌋ (SRS_E2E_08528)

8.2.1.3 E2E_P01ProtectStateType

[SWS_E2E_00020]⌈

Name:	E2E_P01ProtectStateType		
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 the error value. The four high bits are always 0.
Description:	State of the sender for a Data protected with E2E Profile 1.		

⌋ (SRS_E2E_08528)

8.2.1.4 E2E_P01CheckStateType

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

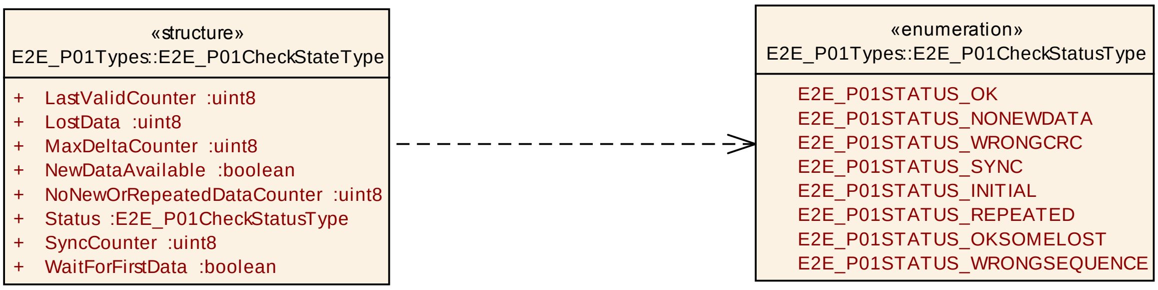


Figure 8-2: E2E Profile 1 check state type

[SWS_E2E_00021]⌈

Name:	E2E_P01CheckStateType		
Type:	Structure		
Element:	uint8	LastValidCounter	Counter value most recently received. If no data has been yet received, then the value is 0x0. After each reception, the counter is updated with the value received.
	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. This attribute is set only if Status equals E2E_P01STATUS_OK or E2E_P01STATUS_OKSOMELOST. For other values of Status, the value of LostData is undefined. E2E_P01CheckStatusType Status Result of the verification of the Data, determined by the Check function.
	E2E_P01CheckStatusType	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 receiver for a Data protected with E2E Profile 1.		

⌋ (SRS_E2E_08528)

8.2.1.5 E2E_P01CheckStatusType

[SWS_E2E_00022]⌈

Name:	E2E_P01CheckStatusType	
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 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P01STATUS_WRONGSEQUENCE	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description:	Result of the verification of the Data in E2E Profile 1, determined by the Check function.	

⌋ (SRS_E2E_08534)

8.2.2 E2E Profile 2 types

8.2.2.1 E2E_P02ConfigType

[SWS_E2E_00152]⌈

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.
	uint16	Offset	Offset of the E2E header in the Data[] array in bits. It shall be: 0 ≤ Offset ≤ MaxDataLength-(2*8).
Description:	Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 2. The position of the counter and CRC is not configurable in profile 2.		

⌋ (SRS_E2E_08528)

8.2.2.2 E2E_P02ProtectStateType

[SWS_E2E_00153]⌈

Name:	E2E_P02ProtectStateType		
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.		

⌋ (SRS_E2E_08528)

8.2.2.3 E2E_P02CheckStateType

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

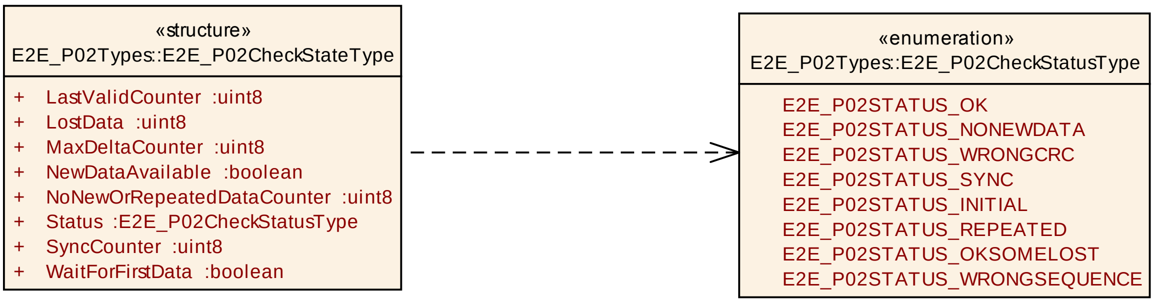


Figure 8-3: E2E Profile 2 check state

[SWS_E2E_00154]⌈

Name:	`E2E_P02CheckStateType`		
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_P02CheckStatusType`	`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.		

⌋ (SRS_E2E_08528)

8.2.2.4 E2E_P02CheckStatusType

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

[SWS_E2E_00214]⌈

Name:	E2E_P02CheckStatusType	
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 DeltaCounter (1 < DeltaCounter =MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P02STATUS_WRONGSEQUENCE	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description:	Result of the verification of the Data in E2E Profile 2, determined by the Check function.	

⌋ (SRS_E2E_08534)

8.2.3 E2E Profile 4 types

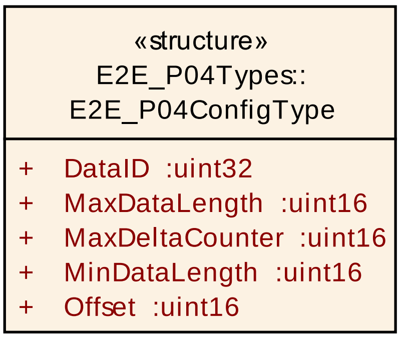


Figure 8-4: E2E Profile 4 configuration

8.2.3.1 E2E_P04ConfigType

[SWS_E2E_00334]⌈

Name:	`E2E_P04ConfigType`		
Type:	`Structure`		
Element:	`uint32`	`DataID`	A system-unique identifier of the Data.
	`uint16`	`Offset`	Bit offset of the first bit of the E2E header from the beginning of the Data Array (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ MaxDataLength-(12*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	`uint16`	`MinDataLength`	Minimal length of Data array, in bits. E2E checks that Length is ≥ MinDataLength. The value shall be = 4096*8 (4kB) and shall be ≥ 12*8
	`uint16`	`MaxDataLength`	Maximal length of Data, in bits. E2E checks that DataLength is ≤ MinDataLength. The value shall be = 4096*8 (4kB) and it shall be ≥ MinDataLength
	`uint16`	`MaxDeltaCounter`	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description:	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 4. For each transmitted Data, there is an instance of this typedef.		

⌋ (SRS_E2E_08539)

8.2.3.2 E2E_P04ProtectStateType

[SWS_E2E_00335]⌈

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

⌋ (SRS_E2E_08539)

8.2.3.3 E2E_P04CheckStateType

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

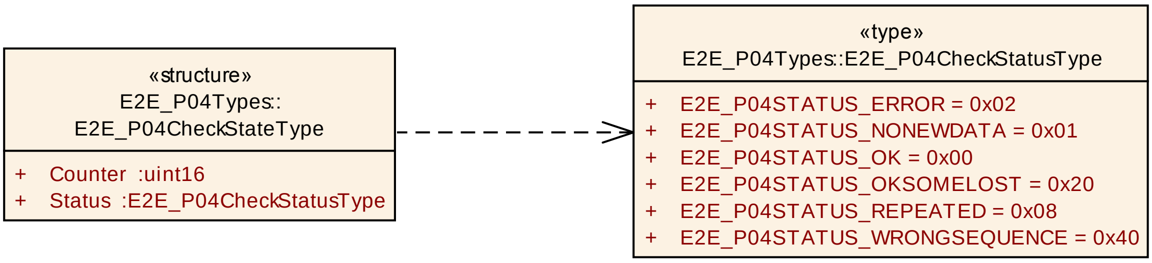


Figure 8-5: E2E Profile 4 check state

[SWS_E2E_00336]⌈

Name:	E2E_P04CheckStateType		
Type:	Structure		
Element:	E2E_P04CheckStatusType	Status	Result of the verification of the Data in this cycle, determined by the Check function.
	uint16	Counter	Counter of the data in previous cycle. It is initialized with 0.
Description:	State of the reception on one single Data protected with E2E Profile 4.		

⌋ (SRS_E2E_08539)

8.2.3.4 E2E_P04CheckStatusType

[SWS_E2E_00337]⌈

Name:	E2E_P04CheckStatusType		
Type:	--		
Range:	E2E_P04STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P04STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P04STATUS_REPEATED.
	E2E_P04STATUS_ERROR	0x02	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P04STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P04STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the

			allowed configured delta).
	E2E_P04STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description:	Status of the reception on one single Data in one cycle, protected with E2E Profile 4.		

⌋ (SRS_E2E_08534)

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

8.2.4 E2E Profile 5 types

8.2.4.1 E2E_P05ConfigType

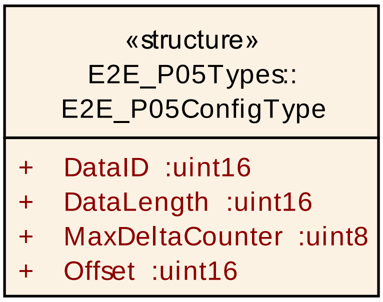


Figure 8-6: E2E Profile 5 configuration

[SWS_E2E_00437]⌈

Name:	`E2E_P05ConfigType`		
Type:	`Structure`		
Element:	`uint16`	`Offset`	Bit offset of the first bit of the E2E header from the beginning of the Data Array (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ DataLength-(3*8). Example: If Offset equals 8, then the low byte of the E2E Crc (16 bit) is written to Byte 1, the high Byte is written to Byte 2.
	`uint16`	`DataLength`	Length of data, in bits. The value shall be = 4096*8 (4kB) and shall be ≥ 3*8
	`uint16`	`DataID`	A system-unique identifier of the Data
	`uint8`	`MaxDeltaCounter`	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description:	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 5. For each transmitted Data, there is an instance of this typedef.		

⌋ (SRS_E2E_08539, SRS_E2E_08534)

8.2.4.2 E2E_P05ProtectStateType

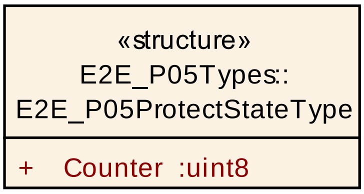


Figure 8-7: E2E Profile 5 Protect state type

[SWS_E2E_00438]⌈

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

⌋ (SRS_E2E_08539)

8.2.4.3 E2E_P05CheckStateType

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

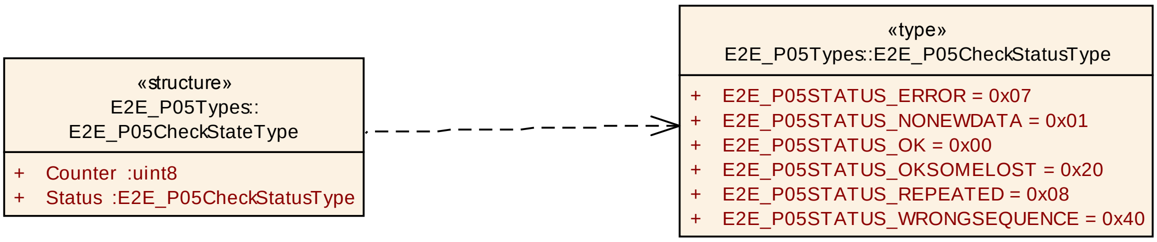


Figure 8-8: E2E Profile 5 Check state type

[SWS_E2E_00439]⌈

Name:	E2E_P05CheckStateType		
Type:	Structure		
Element:	E2E_P05CheckStatusType	Status	Result of the verification of the Data in this cycle, determined by the Check function.
	uint8	Counter	Counter of the data in previous cycle. It is initialized with 0.
Description:	Description: State of the reception on one single Data protected with E2E Profile 5.		

⌋ (SRS_E2E_08539)

8.2.4.4 E2E_P05CheckStatusType

[SWS_E2E_00440]⌈

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

			counter jump, which changed more than the allowed delta
Description:	Status of the reception on one single Data in one cycle, protected with E2E Profile 5.		

⌋ (SRS_E2E_08539)

8.2.5 E2E Profile 6 types

8.2.5.1 E2E_P06ConfigType

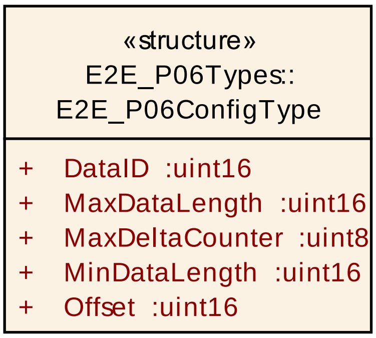


Figure 8-9: E2E Profile 6 configuration

[SWS_E2E_00441]⌈

Name:	E2E_P06ConfigType		
Type:	Structure		
Element:	uint16	Offset	Bit offset of the first bit of the E2E header from the beginning of the Data Array (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ MaxDataLength-(5*8). Example: If Offset equals 8, then the high byte of the E2E Crc (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	uint16	MinDataLength	Minimal length of Data array, in bytes (i.e. the length of the E2E-protected message, without counting the field Length itself). E2E checks that Length is ≥ MinDataLength. The value shall be = 4096*8 (4kB) and shall be ≥ 5*8
	uint16	MaxDataLength	Maximal length of Data, in bits. E2E checks that DataLength is ≤ MaxDataLength. The value shall be = 4096*8 (4kB). MaxDataLength shall be ≥ MinDataLength
	uint16	DataID	A system-unique identifier of the Data
	uint8	MaxDeltaCounter	Maximum allowed gap between two

			counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description:	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 6. For each transmitted Data, there is an instance of this typedef.		

⌋ (SRS_E2E_08539, SRS_E2E_08534)

8.2.5.2 E2E_P06ProtectStateType

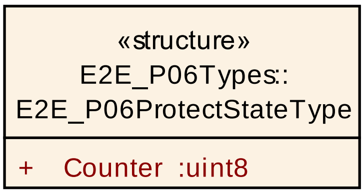


Figure 8-10: E2E Profile 6 Protect state type

[SWS_E2E_00443]⌈

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

⌋ (SRS_E2E_08539)

8.2.5.3 E2E_P06CheckStateType

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

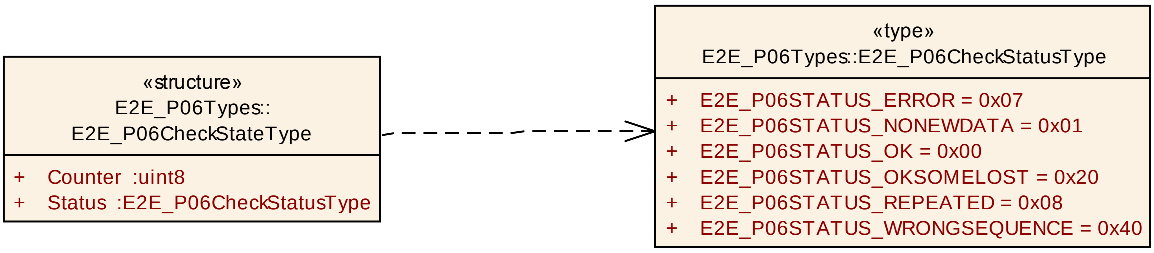


Figure 8-11: E2E Profile 6 Check state type

[SWS_E2E_00444]⌈

Name:	E2E_P06CheckStateType		
Type:	Structure		
Element:	E2E_P06CheckStatusType	Status	Result of the verification of the Data in this cycle, determined by the Check function.
	uint8	Counter	Counter of the data in previous cycle. It is initialized with 0.
Description:	State of the reception on one single Data protected with E2E Profile 6.		

⌋ (SRS_E2E_08539)

8.2.5.4 E2E_P06CheckStatusType

[SWS_E2E_00445]⌈

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

⌋ (SRS_E2E_08539)

8.2.6 E2E state machine types

8.2.6.1 E2E_PCheckStatusType

[SWS_E2E_00347]⌈

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

⌋ (SRS_E2E_08539)

8.2.6.2 E2E_SMConfigType

[SWS_E2E_00342]⌈

Name:	E2E_SMConfigType		
Type:	Structure		
Element:	uint8	WindowSize	Size of the monitoring window for the state machine.
	uint8	MinOkStateInit	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_INIT) required to change to state E2E_SM_VALID.
	uint8	MaxErrorStateInit	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INIT).
	uint8	MinOkStateValid	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_VALID) required to keep in state E2E_SM_VALID.
	uint8	MaxErrorStateValid	Maximal number of checks in which

			ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_VALID).
	uint8	MinOkStateInvalid	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_INVALID) required to change to state E2E_SM_VALID.
	uint8	MaxErrorStateInvalid	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INVALID).
Description:	Configuration of a communication channel for exchanging Data.		

⌋ (SRS_E2E_08539)

8.2.6.3 E2E_SMCheckStateType

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

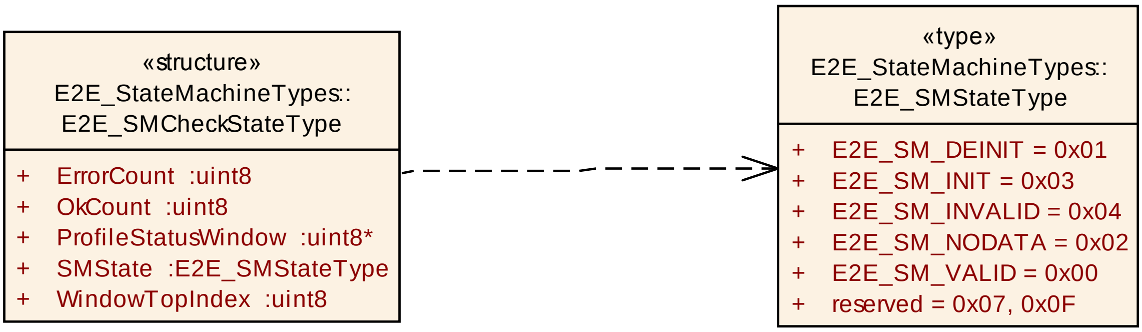


Figure 8-12: E2E SM check state

[SWS_E2E_00343]⌈

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

			within the last WindowSize checks.
	E2E_SMStateType	SMState	The current state in the state machine. The value is not explicitly used in the pseudocode of the state machine, because it is expressed in UML as UML states.
Description:	State of the protection of a communication channel.		

⌋ (SRS_E2E_08539)

8.2.6.4 E2E_SMStateType

[SWS_E2E_00344]⌈

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

⌋ (SRS_E2E_08539)

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 8.3.7, 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]⌈

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

⌋ (SRS_E2E_08528)

8.3.1.2 E2E_P01ProtectInit

[SWS_E2E_00385]⌈

Service name:	E2E_P01ProtectInit	
Syntax:	`Std_ReturnType E2E_P01ProtectInit( E2E_P01ProtectStateType* StatePtr )`	
Service ID[hex]:	0x1b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	

⌋ (SRS_E2E_08528)

[SWS_E2E_00386]⌈ In case State is NULL, E2E_P01ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.⌋ (SRS_E2E_08528)

8.3.1.3 E2E_P01Check

[SWS_E2E_00158]⌈

Service name:	E2E_P01Check	
Syntax:	`Std_ReturnType E2E_P01Check( E2E_P01ConfigType* Config, E2E_P01CheckStateType* 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 SWS_E2E_00047.
Description:	Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.	

⌋ (SRS_E2E_08528)

8.3.1.4 E2E_P01CheckInit

[SWS_E2E_00390]⌈

Service name:	E2E_P01CheckInit	
Syntax:	`Std_ReturnType E2E_P01CheckInit( E2E_P01CheckStateType* State )`	
Service ID[hex]:	0x1c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	State	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the check state	

⌋ (SRS_E2E_08528)

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

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

5. LostData = 0
6. Status = E2E_PXXSTATUS_NONEWDATA
7. NoNewOrRepeatedDataCounter = 0
8. SyncCounter = 0.⌋ (SRS_E2E_08528)

8.3.1.5 E2E_P01MapStatusToSM

[SWS_E2E_00382]⌈

Service name:	E2E_P01MapStatusToSM	
Syntax:	`E2E_PCheckStatusType E2E_P01MapStatusToSM( Std_ReturnType CheckReturn, E2E_P01CheckStatusType Status, boolean profileBehavior )`	
Service ID[hex]:	0x1d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CheckReturn	Return value of the E2E_P01Check function
	Status	Status determined by E2E_P01Check function
	profileBehavior	FALSE: check has the legacy behavior, before R4.2 TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2
Parameters (inout):	None	
Parameters (out):	None	
Return value:	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description:	The function maps the check status of Profile 1 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 1 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	

⌋ (SRS_E2E_08528)

This represents the R4.2 behavior:

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

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

⌋ (SRS_E2E_08528)

This represents the pre-R4.2 behavior:

[SWS_E2E_00476]⌈

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

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

⌋ (SRS_E2E_08528)

[SWS_E2E_00384]⌈ If CheckReturn != E2E_E_OK, then the function E2E_P01MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋ (SRS_E2E_08528)

8.3.2 E2E Profile 2 routines

8.3.2.1 E2E_P02Protect

[SWS_E2E_00160]⌈

Service name:	E2E_P02Protect	
Syntax:	`Std_ReturnType E2E_P02Protect( E2E_P02ConfigType* ConfigPtr, E2E_P02ProtectStateType* StatePtr, uint8* DataPtr )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
Parameters (inout):	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to the data to be protected.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.	

⌋ (SRS_E2E_08528)

8.3.2.2 E2E_P02ProtectInit

[SWS_E2E_00387]⌈

Service name:	E2E_P02ProtectInit
Syntax:	`Std_ReturnType E2E_P02ProtectInit( E2E_P02ProtectStateType* StatePtr`

	)	
Service ID[hex]:	0x1e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	

⌋ (SRS_E2E_08528)

[SWS_E2E_00388]⌈ In case State is NULL, E2E_P02ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.⌋ (SRS_E2E_08528)

8.3.2.3 E2E_P02Check

[SWS_E2E_00161]⌈

Service name:	E2E_P02Check	
Syntax:	`Std_ReturnType E2E_P02Check( E2E_P02ConfigType* ConfigPtr, E2E_P02CheckStateType* StatePtr, uint8* DataPtr )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	DataPtr	--
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.	

⌋ (SRS_E2E_08528)

8.3.2.4 E2E_P02CheckInit

[SWS_E2E_00391]⌈

Service name:	E2E_P02CheckInit
Syntax:	`Std_ReturnType E2E_P02CheckInit( E2E_P02CheckStateType* StatePtr )`
Service ID[hex]:	0x1f

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the check state	

⌋ (SRS_E2E_08528)

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

1. LastValidCounter = 0
2. MaxDeltaCounter = 0
3. WaitForFirstData = FALSE
4. NewDataAvailable = TRUE
5. LostData = 0
6. Status = E2E_PXXSTATUS_NONEWDATA
7. NoNewOrRepeatedDataCounter = 0
8. SyncCounter = 0.⌋ (SRS_E2E_08528)

8.3.2.5 E2E_P02MapStatusToSM

[SWS_E2E_00379]⌈

Service name:	E2E_P02MapStatusToSM	
Syntax:	`E2E_PCheckStatusType E2E_P02MapStatusToSM( Std_ReturnType CheckReturn, E2E_P02CheckStatusType Status, boolean profileBehavior )`	
Service ID[hex]:	0x20	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CheckReturn	Return value of the E2E_P02Check function
	Status	Status determined by E2E_P02Check function
	profileBehavior	FALSE: check has the legacy behavior, before R4.2 TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2
Parameters (inout):	None	
Parameters (out):	None	
Return value:	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description:	The function maps the check status of Profile 2 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 2 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	

⌋ (SRS_E2E_08528)

This represents the R4.2 behavior:

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

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

⌋ (SRS_E2E_08528)

This represents the pre-R4.2 behavior:
[SWS_E2E_00477]⌈
If CheckReturn == E2E_E_OK and ProfileBehavior == 0, then the function E2E_P02MapStatusToSM shall return the values depending on the value of Status:

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

⌋ (SRS_E2E_08528)

[SWS_E2E_00381]⌈ If CheckReturn != E2E_E_OK, then the function E2E_P02MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋ (SRS_E2E_08528)

8.3.3 E2E Profile 4 routines

8.3.3.1 E2E_P04Protect

[SWS_E2E_00338]⌈

Service name:	E2E_P04Protect	
Syntax:	`Std_ReturnType E2E_P04Protect( E2E_P04ConfigType* ConfigPtr, E2E_P04ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length )`	
Service ID[hex]:	0x21	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.

Parameters (inout):	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 4. This includes checksum calculation, handling of counter and Data ID.	

⌋ (SRS_E2E_08539)

8.3.3.2 E2E_P04ProtectInit

[SWS_E2E_00373]⌈

Service name:	E2E_P04ProtectInit	
Syntax:	`Std_ReturnType E2E_P04ProtectInit( E2E_P04ProtectStateType* StatePtr )`	
Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	

⌋ (SRS_E2E_08539)

[SWS_E2E_00377]⌈ In case State is NULL, E2E_P04ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.⌋ (SRS_E2E_08539)

8.3.3.3 E2E_P04Check

[SWS_E2E_00339]⌈

Service name:	E2E_P04Check	
Syntax:	`Std_ReturnType E2E_P04Check( E2E_P04ConfigType* ConfigPtr, E2E_P04CheckStateType* StatePtr, uint8* DataPtr, uint16 Length )`	
Service ID[hex]:	0x23	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout):	StatePtr	Pointer to received data.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Checks the Data received using the E2E profile 4. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	

⌋ (SRS_E2E_08539)

8.3.3.4 E2E_P04CheckInit

[SWS_E2E_00350]⌈

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

⌋ (SRS_E2E_08539)

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

1. Counter to 0
2. Status to E2E_P04STATUS_ERROR.⌋ (SRS_E2E_08539)

8.3.3.5 E2E_P04MapStatusToSM

[SWS_E2E_00349]⌈

Service name:	E2E_P04MapStatusToSM	
Syntax:	`E2E_PCheckStatusType E2E_P04MapStatusToSM( Std_ReturnType CheckReturn, E2E_P04CheckStatusType Status )`	
Service ID[hex]:	0x25	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CheckReturn	Return value of the E2E_P04Check function
	Status	Status determined by E2E_P04Check function
Parameters (inout):	None	
Parameters (out):	None	
Return value:	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description:	The function maps the check status of Profile 4 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 4 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	

⌋ (SRS_E2E_08539)

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

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

⌋ (SRS_E2E_08539)

[SWS_E2E_00352]⌈ If CheckReturn != E2E_E_OK, then the function E2E_P04MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋ (SRS_E2E_08539)

8.3.4 E2E Profile 5 routines

8.3.4.1 E2E_P05Protect

[SWS_E2E_00446]⌈

Service name:	E2E_P05Protect

Syntax:	`Std_ReturnType E2E_P05Protect( E2E_P05ConfigType* ConfigPtr, E2E_P05ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length )`	
Service ID[hex]:	0x26	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout):	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 5. This includes checksum calculation, handling of counter.	

⌋ (SRS_E2E_08539)

8.3.4.2 E2E_P05ProtectInit

[SWS_E2E_00447]⌈

Service name:	E2E_P05ProtectInit	
Syntax:	`Std_ReturnType E2E_P05ProtectInit( E2E_P05ProtectStateType* StatePtr )`	
Service ID[hex]:	0x27	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	

⌋ (SRS_E2E_08539)

[SWS_E2E_00448]⌈ In case State is NULL, E2E_P05ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.⌋ (SRS_E2E_08539)

8.3.4.3 E2E_P05Check

[SWS_E2E_00449]⌈

Service name:	E2E_P05Check	
Syntax:	`Std_ReturnType E2E_P05Check( E2E_P05ConfigType* ConfigPtr, E2E_P05CheckStateType* StatePtr, uint8* DataPtr, uint16 Length )`	
Service ID[hex]:	0x28	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Checks the Data received using the E2E profile 5. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	

⌋ (SRS_E2E_08539)

8.3.4.4 E2E_P05CheckInit

[SWS_E2E_00450]⌈

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

⌋ (SRS_E2E_08539)

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

1. Counter to 0
2. Status to E2E_P05STATUS_ERROR.⌋ (SRS_E2E_08539)

8.3.4.5 E2E_P05MapStatusToSM

[SWS_E2E_00452]⌈

Service name:	E2E_P05MapStatusToSM	
Syntax:	`E2E_PCheckStatusType E2E_P05MapStatusToSM( Std_ReturnType CheckReturn, E2E_P05CheckStatusType Status )`	
Service ID[hex]:	0x2a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CheckReturn	Return value of the E2E_P05Check function
	Status	Status determined by E2E_P05Check function
Parameters (inout):	None	
Parameters (out):	None	
Return value:	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description:	The function maps the check status of Profile 5 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 5 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	

⌋ (SRS_E2E_08539)

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

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

⌋ (SRS_E2E_08539)

[SWS_E2E_00454]⌈ If CheckReturn != E2E_E_OK, then the function E2E_P05MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋ (SRS_E2E_08539)

8.3.5 E2E Profile 6 routines

8.3.5.1 E2E_P06Protect

[SWS_E2E_00393]⌈

Service name:	E2E_P06Protect

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

⌋ (SRS_E2E_08539)

8.3.5.2 E2E_P06ProtectInit

[SWS_E2E_00455]⌈

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

⌋ (SRS_E2E_08539)

[SWS_E2E_00456]⌈ In case State is NULL, E2E_P06ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.⌋ (SRS_E2E_08539)

8.3.5.3 E2E_P06Check

[SWS_E2E_00457]⌈

Service name:	E2E_P06Check	
Syntax:	`Std_ReturnType E2E_P06Check( E2E_P06ConfigType* ConfigPtr, E2E_P06CheckStateType* StatePtr, uint8* DataPtr, uint16 Length )`	
Service ID[hex]:	0x2d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Checks the Data received using the E2E profile 6. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	

⌋ (SRS_E2E_08539)

8.3.5.4 E2E_P06CheckInit

[SWS_E2E_00458]⌈

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

⌋ (SRS_E2E_08539)

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

1. Counter to 0
2. Status to E2E_P06STATUS_ERROR.⌋ (SRS_E2E_08539)

8.3.5.5 E2E_P06MapStatusToSM

[SWS_E2E_00460]⌈

Service name:	E2E_P06MapStatusToSM	
Syntax:	`E2E_PCheckStatusType E2E_P06MapStatusToSM( Std_ReturnType CheckReturn, E2E_P06CheckStatusType Status )`	
Service ID[hex]:	0x2f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CheckReturn	Return value of the E2E_P06Check function
	Status	Status determined by E2E_P06Check function
Parameters (inout):	None	
Parameters (out):	None	
Return value:	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description:	The function maps the check status of Profile 6 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 6 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	

⌋ (SRS_E2E_08539)

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

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

⌋ (SRS_E2E_08539)

[SWS_E2E_00462]⌈ If CheckReturn != E2E_E_OK, then the function E2E_P06MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋ (SRS_E2E_08539)

8.3.6 E2E State machine routines

8.3.6.1 E2E_SMCheck

[SWS_E2E_00340]⌈

Service name:	E2E_SMCheck

Syntax:	`Std_ReturnType E2E_SMCheck( E2E_PCheckStatusType ProfileStatus, E2E_SMConfigType* ConfigPtr, E2E_SMCheckStateType* StatePtr )`	
Service ID[hex]:	0x30	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ProfileStatus	Profile-independent status of the reception on one single Data in one cycle
	ConfigPtr	Pointer to static configuration.
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK E2E_E_WRONGSTATE For definitions for return values, see SWS_E2E_00047.
Description:	Checks the communication channel. It determines if the data can be used for safety-related application, based on history of checks performed by a corresponding E2E_P0XCheck() function.	

⌋ (SRS_E2E_08539)

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

Else, the function E2E_SMCheck shall perform the logic according to the specified state machine.⌋ (SRS_E2E_08539)

8.3.6.2 E2E_SMCheckInit

[SWS_E2E_00353]⌈

Service name:	E2E_SMCheckInit	
Syntax:	`Std_ReturnType E2E_SMCheckInit( E2E_SMCheckStateType* StatePtr, E2E_SMConfigType* ConfigPtr )`	
Service ID[hex]:	0x31	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to configuration of the state machine
Parameters (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the state machine.	

⌋ (SRS_E2E_08539)

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

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

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

and it shall return with E2E_E_OK.⌋ (SRS_E2E_08539)

8.3.7 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.

[SWS_E2E_00106] {OBSOLETE}⌈ When calculating a CRC8 by calling any E2E_CRC8_* function (single call), the caller shall use the protocol-specific start value as E2E_StartValue.⌋ (SRS_E2E_08528)

[SWS_E2E_00107] {OBSOLETE}⌈ When calculating a resulting CRC by multiple calls, the first call shall use the protocol-specific start value as E2E_StartValue. For the subsequent calls, the caller shall generate E2E_StartValue like follows: the result of previous E2E_CRC8*() call XOR-ed with protocol-specific XOR value. ⌋ (SRS_E2E_08528)

The below code example illustrates the above requirements (note that XORing with 0x00 makes little sense, but it makes sense for other values like 0xFF):

```
/* buffer to protect */
uint16 Data[30] = {...};

uint8 i = 0;

/* first step - start with 0x00 */
uint8 CRC = E2E_CRC8u16(Data[0], 0x00);

for(i = 1; i < 30; i++) {
```

```
      /* ith step: start based on previous CRC^0x00 */
      CRC = E2E_CRC8u16(Data[i], CRC ^ 00);
}
```

The elementary functions are defined by groups. For a compact representation, the data types mnemonics are used. For example, there is a CRC function for several data types, and for each data type of the parameter <InType>, there is a different function suffix <InTypeMn>.

Size	Platform Type <InType>	Mnemonic <InTypeMn>
unsigned 8-Bit	uint8	u8
unsigned 16-Bit	uint16	u16
unsigned 32-Bit	uint32	u32

Table 8-1: Types and mnemonics for template routines

8.3.7.1 E2E_CRC8<InTypeMn>

[SWS_E2E_00092] {OBSOLETE}⌈

Service name:	E2E_CRC8<InTypeMn> (obsolete)	
Syntax:	uint8 E2E_CRC8<InTypeMn>(<InType> Data, uint8 StartValue)	
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:	InTypeMn: {u8, u16, u32}, which is the one corresponding to InType. 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. Relation to Crc_CalculateCRC8(): E2E_CRC8_*() may simply call Crc_CalculateCRC8() in a loop. The function uses SAE J1850 polynomial, but with 0x00 as start value and XOR value. **Tags:** atp.Status=obsolete	

	atp.StatusRevisionBegin=4.2.2

⌋ (SRS_E2E_08528)

[SWS_E2E_00091]{OBSOLETE}⌈

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)

⌋ (SRS_E2E_08528)

8.3.7.2 E2E_CRC8<InTypeMn>Array

[SWS_E2E_00094] {OBSOLETE}⌈

Service name:	E2E_CRC8<InTypeMn>Array (obsolete)	
Syntax:	`uint8 E2E_CRC8<InTypeMn>Array( const <InType>* E2E_DataPtr, uint32 ArrayLength, uint8 StartValue )`	
Service ID[hex]:	0x0A, 0x0B, 0x0C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	E2E_DataPtr	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	ArrayLength	Length of array (data block) to be calculated in bytes.
	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	CRC 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. **Tags:** atp.Status=obsolete atp.StatusRevisionBegin=4.2.2	

⌋ (SRS_E2E_08528)

[SWS_E2E_00095] {OBSOLETE}⌈

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)

⌋ (SRS_E2E_08528)

8.3.7.3 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] {OBSOLETE}⌈

Service name:	E2E_CRC8H2F<InTypeMn> (obsolete)	
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:	InTypeMn: {u8, u16, u32}, which is the one corresponding to InType. 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. **Tags:** atp.Status=obsolete atp.StatusRevisionBegin=4.2.2	

⌋ (SRS_E2E_08528)

[SWS_E2E_00276] {OBSOLETE}⌈

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)

⌋ (SRS_E2E_08528)

8.3.7.4 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] {OBSOLETE}⌈

Service name:	E2E_CRC8H2F<InTypeMn>Array (obsolete)	
Syntax:	`uint8 E2E_CRC8H2F<InTypeMn>Array( const <InType>* E2E_DataPtr, uint32 ArrayLength, uint8 StartValue )`	
Service ID[hex]:	0x10, 0x11, 0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	E2E_DataPtr	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	ArrayLength	Length of array (data block) to be calculated in bytes.
	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 arrary 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. **Tags:** atp.Status=obsolete atp.StatusRevisionBegin=4.2.2	

⌋ (SRS_E2E_08527)

[SWS_E2E_00098] {OBSOLETE}⌈

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)

⌋ (SRS_E2E_08528)

8.3.7.5 E2E_UpdateCounter

[SWS_E2E_00099] {OBSOLETE}⌈

Service name:	E2E_UpdateCounter (obsolete)	
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. **Tags:** atp.Status=obsolete atp.StatusRevisionBegin=4.2.2	

⌋ (SRS_E2E_08528)

8.3.8 Auxiliary Functions

8.3.8.1 E2E_GetVersionInfo

[SWS_E2E_00032]⌈

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

⌋ (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⌋ (SRS_E2E_08528)

8.4 Call-back notifications

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

8.5 Scheduled functions

None. The E2E library does not have scheduled functions.

8.6 Expected Interfaces

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

[SWS_E2E_00110]⌈ The E2E library shall not call any functions from external modules apart from explicitly listed expected interfaces of E2E Library. ⌋ (SRS_E2E_08528)

8.6.1 Mandatory Interfaces

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

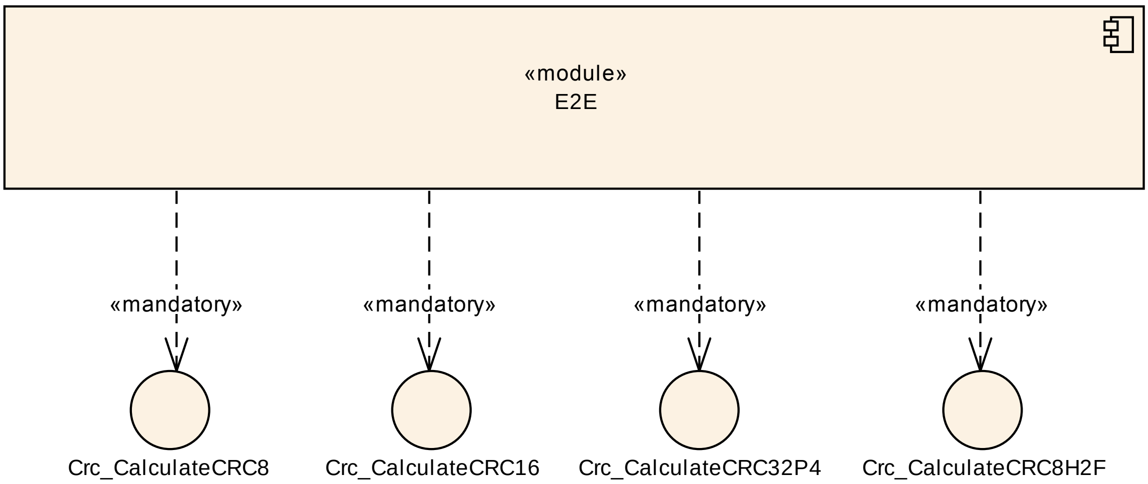


Figure 8-13: Expected mandatory interfaces by E2E library

9 Sequence Diagrams for invoking E2E Library

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

9.1 Sender

[UC_E2E_00202]⌈ During its initialization, the Sender shall instantiate the structures PXXConfigType and PXXProtectStateType, separately for each Data to be protected.
⌋ (SRS_E2E_08528)

[UC_E2E_00203]⌈ During its initialization, the Sender shall initialize the PXXConfigType with the required configured settings, for each Data to be protected.
⌋ (SRS_E2E_08528)

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

[UC_E2E_00204]⌈ During its initialization, the Sender shall initialize the E2E_PXXProtectStateType for each Data, with the configured following values: Counter = 0.⌋ (SRS_E2E_08528)

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

This means that is not allowed e.g. to call E2E_PXXProtect() twice without having Rte_Send_<p>_<o>() in between. It is also not allowed e.g. to call PduR_ComTransmit() twice without having E2E_PXXProtect() in between.⌋ (SRS_E2E_08528)

9.1.1 Sender of data elements

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

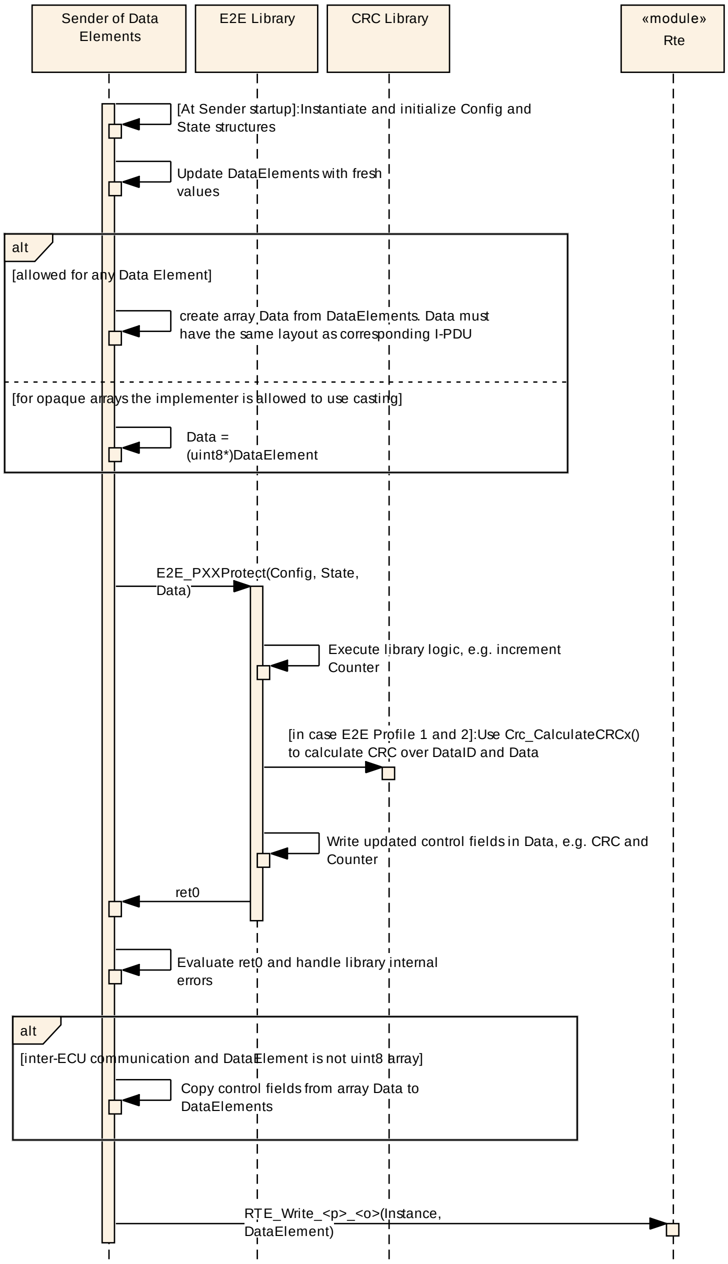


Figure 9-1: Sender of data elements

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

[UC_E2E_00230]⌈ In case the data element communication is inter-ECU and the data element is not an opaque uint8 array, then the user of the E2E Library shall serialize the data element into the array Data. The content of the array Data shall be the equal to the content of the serialized representation of corresponding signal group in an I-PDU.⌋ (SRS_E2E_08528)

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

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

[UC_E2E_00232]⌈ For sending of data elements different from opaque arrays, the caller of E2E Library shall serialize the data element to Data, then it shall call the E2E_PXXProtect() routine and then it shall copy back the control fields from Data to data element.⌋ (SRS_E2E_08528)

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

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

9.1.2 Sender at sigal group level

The diagram below species the overall sequence involving the E2E Library by the Sender at the signal group level. The Sender itself can be realized by one or more modules/files (e.g. COM plus callouts, or COM plus complex device driver).
The diagram shows the example when there is only one E2E-protected signal group in the I-PDU, but in general it is possible to have several of them (0 or 1 E2E-protections per signal group). In such case, the sender of I-PDUs invokes E2E_PXXProtect on each E2E-protected signal group.

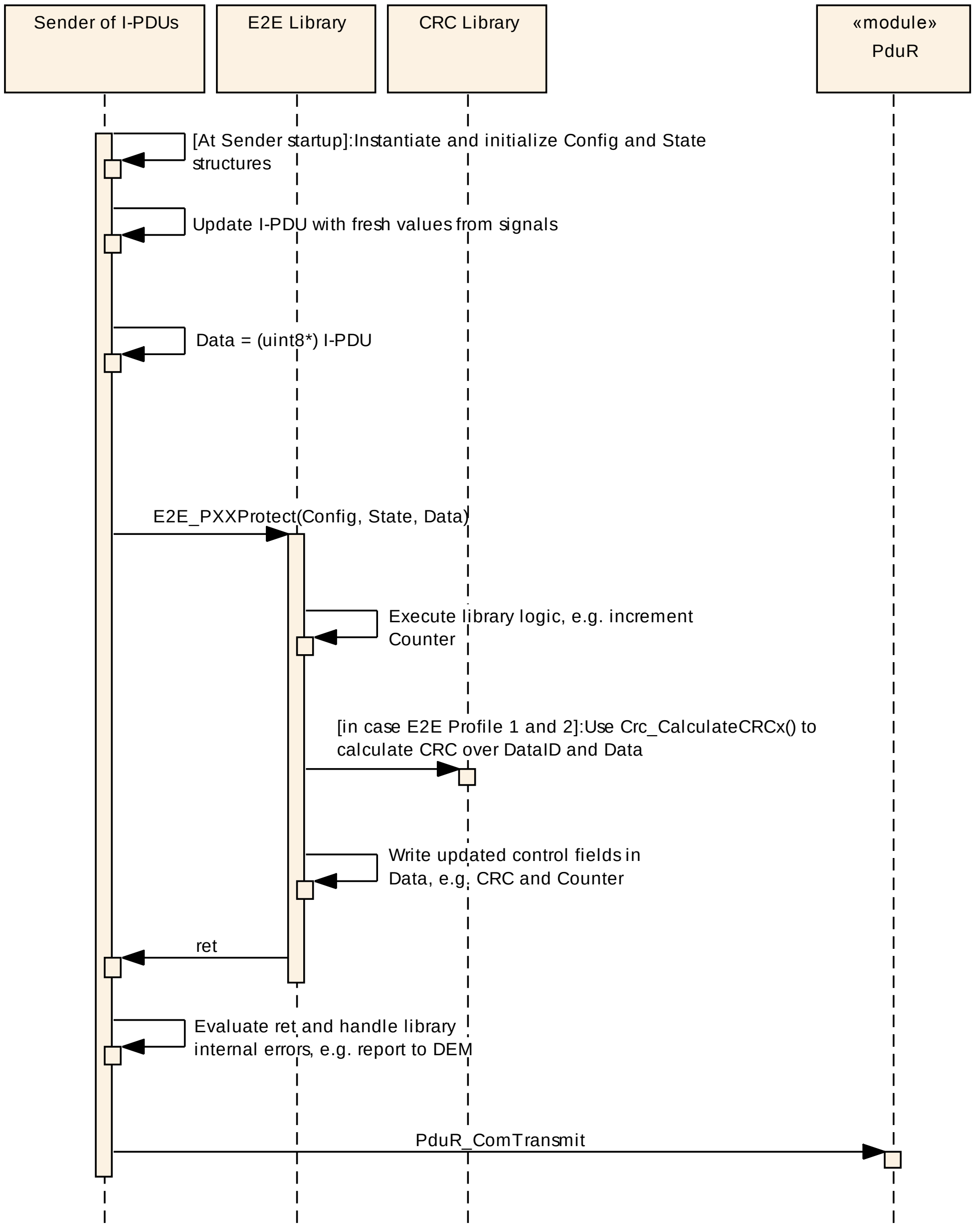


Figure 9-2: Sender of I-PDUs

9.2 Receiver

[UC_E2E_00206]⌈ During its initialization, the Receiver shall instantiate the structures PXXConfigType and PXXReceiverType.⌋ (SRS_E2E_08528)

Note: When selecting the following initialization and configuration parameters the functional behaviour of the enhanced E2E_PXXCheck()-functions (introduced in AUTOSAR R4.0.4 and R3.2.2) is application-wise backward compatible to the E2E_PxxCheck()-function of the earlier AUTOSAR releases:

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

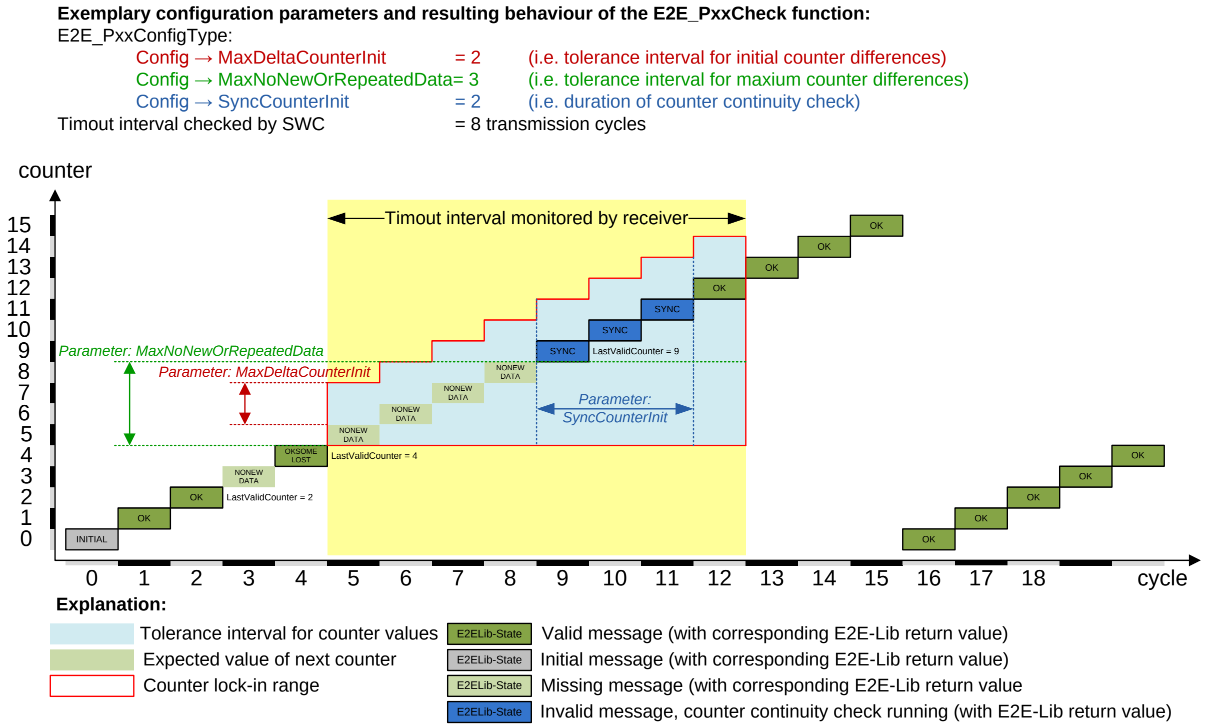


Figure 9-3: Configuration parameters of the E2E_PxxCheck() function and their effects

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

[UC_E2E_00207]⌈ During its initialization, the Receiver shall initialize the PXXConfigType with the required configured settings, for each Data. ⌋ (SRS_E2E_08528)

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

[UC_E2E_00208]⌈ During its initialization, the Receiver shall initialize the E2E_PXXCheckStateType with the following values:

LastValidCounter = 0
MaxDeltaCounter = 0
SyncCounter = 0
NoNewOrRepeatedDataCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0

Status = E2E_PXXSTATUS_NONEWDATA⌋ (SRS_E2E_08528)

[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().⌋ (SRS_E2E_08528)

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

[UC_E2E_00233]⌈ If the return from the function E2E_PXXCheck() is different than E2E_PXXSTATUS_OK and E2E_PXXSTATUS_OKSOMELOST, then the caller shall not evaluate the attribute State->LostData.⌋ (SRS_E2E_08528)

9.2.1 Receiver at data element level

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

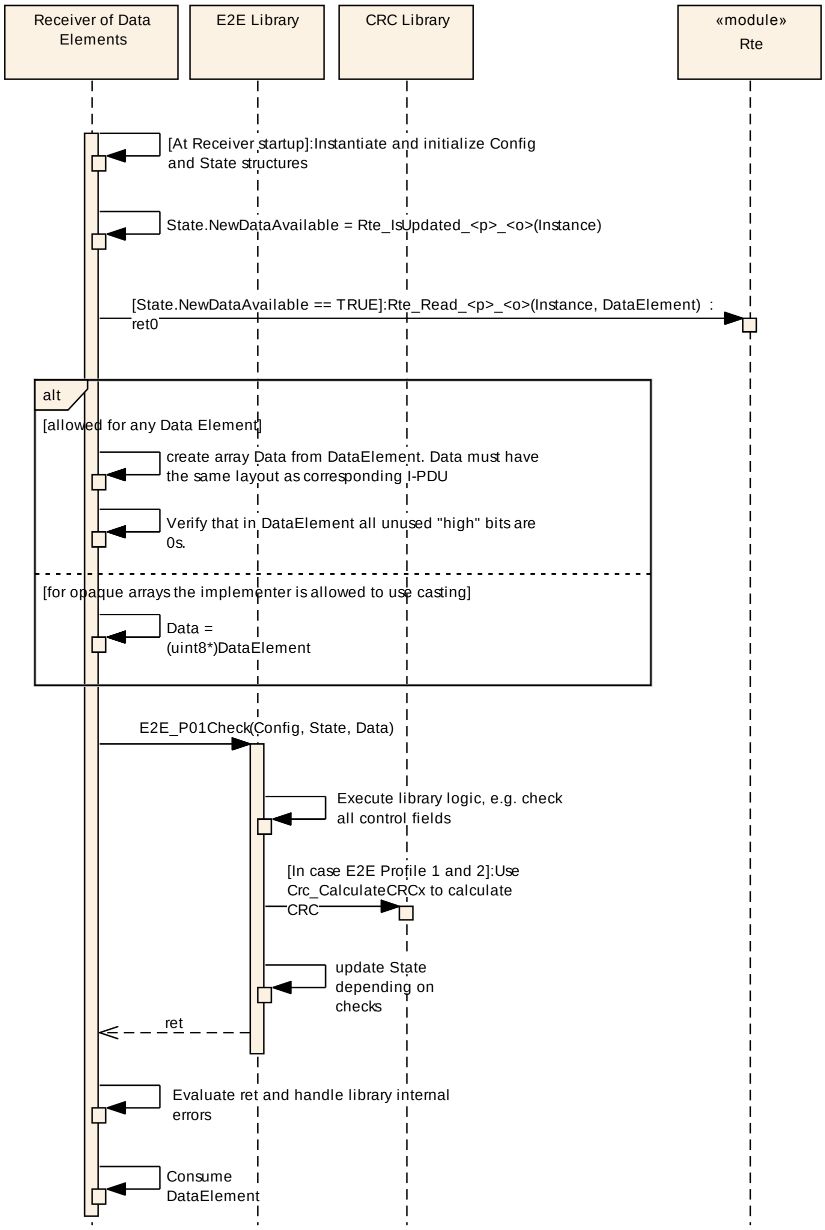


Figure 9-4: Receiver of data elements

[UC_E2E_00277]⌈ In case the data element communication is inter-ECU and the data element is not an opaque uint8 array, then the Receiver shall serialize the data element into the array Data. The layout (content) of Data shall be the same as the layout of the corresponding I-PDU over which the data element is sent. Moreover, the Receiver shall also verify that all bits that are not transmitted in I-PDU (i.e. which are not present in Data) are equal to 0.⌋ (SRS_E2E_08528)

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

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

[UC_E2E_00278]⌈ For reception of data elements different from opaque arrays, the caller of E2E Library shall serialize the data element to Data, then it shall call the check routine.⌋ (SRS_E2E_08528)

9.2.2 Receiver at signal group level

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

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

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

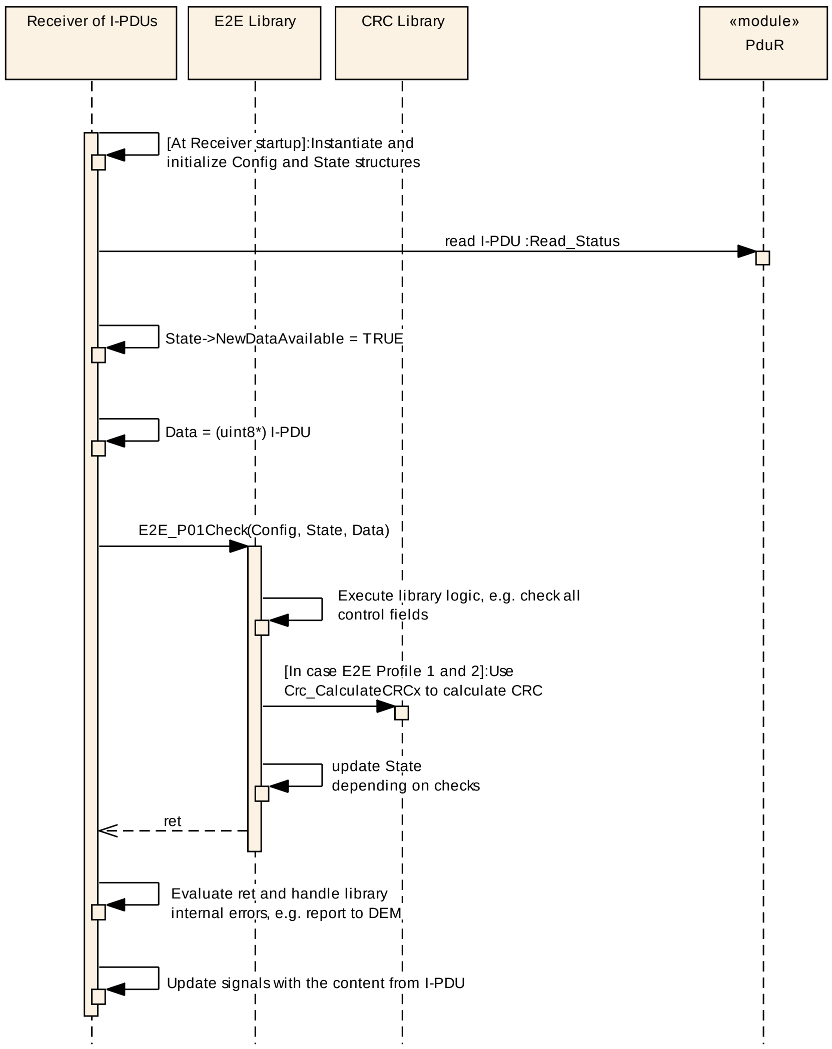


Figure 9-5: Receiver of I-PDUs

10 Configuration specification

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

[SWS_E2E_00037]⌈ The E2E library shall not have any configuration options.⌋ (SRS_BSW_00344, SRS_BSW_00345, SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00171, SRS_BSW_00170, SRS_BSW_00101)

10.1 Published Information

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

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

11 Annex A: Safety Manual for usage of E2E Library

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

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

11.1 E2E profiles and their standard variants

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

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

[UC_E2E_00053]⌈ Any user of E2E Profile 1 shall use whenever possible the defined E2E variants.⌋ (SRS_E2E_08528)

11.2 E2E error handling

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

A general standardization of the error handing 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.⌋
(SRS_E2E_08528)

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
E2E Profile 4	4 kB

Figure 11-1: Maximum lengths

In E2E Profiles 1 and 2, 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, 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.⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08528)

[UC_E2E_00315]⌈ In case of inter-ECU, the length of the complete Data (including application data and E2E header) protected by E2E Profile 4 shall not exceed 4kB.⌋ (SRS_E2E_08528)

The requirements UC_E2E_00051, UC_E2E_00061 and UC_E2E_00315 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.⌋ (SRS_E2E_08539)

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.⌋ (SRS_E2E_08528, SRS_E2E_08537)

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 11.9.

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 Profiles 1, 4, 5, 6) 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 (E2E Profiles 1, 4, 5, 6) or a unique DataIDList[] (for E2e Profile 2).⌋ (SRS_E2E_08528)

[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 (E2E Profiles 1, 4, 5, 6) or a unique DataIDList[] (Profile 2).⌋ (SRS_E2E_08528)

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)} ).
```

⌋ (SRS_E2E_08528)

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)} ).
```

⌋ (SRS_E2E_08528)

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 incluced 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]⌈ Any user of Profile 1 in Nibble Data ID configuration shall ensure that:

1. 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.⌋ (SRS_E2E_08528)

[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 != 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.⌋ (SRS_E2E_08527)

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.⌋ (SRS_E2E_08528)

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]⌈ When using E2E Profiles, 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.⌋ (SRS_E2E_08528)

[UC_E2E_00063]⌈ When using E2E Profiles, signals that have length >= 8 bits should start or finish at the byte limit of an I-PDU.⌋ (SRS_E2E_08528)

[UC_E2E_00320]⌈ When using E2E Profiles, the length of the data to be protected shall be multiple of 8 bits.⌋ (SRS_E2E_08528)

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 libary 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 (sytem template parameter ISignalIPdu.unusedBitPattern). ⌋ (SRS_E2E_08528)

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.

[UC_E2E_00465]⌈ In case E2E Library is invoked by E2E Transformer, then the serializer transformer shall set all unused bits/bytes, if any, to any determined/deterministic value.
⌋ (SRS_E2E_08528)

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
TCP/IP	Big 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 that have been initially targeted by E2E, which have been FlexRay, CAN and LIN are Little Endian, which results with the following requirement:

[UC_E2E_00055]⌈ Any user of E2E Profile 1, 2 and 5 shall place multibyte data in Little Endian order.⌋ (SRS_E2E_08528)

However, the TCP/IP stack is Big Endian. The E2E Profile 4 and 6 can be used for FlexRay TP and CAN TP, but the main use case is TCP/IP. Moreover, TCP/IP can be considered as more future ordiented, therefore Big Endian is foreseen for E2E Profile 4 and 6:

[UC_E2E_00316]⌈ Any user of E2E Profile 4 and 6 shall place multibyte data in Big Endian order.⌋ (SRS_E2E_08539)

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.⌋ (SRS_E2E_08528)

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 3rd stored by microcontroller as 3rd from “left” (for LSB first) or 3rd 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
Ethernet	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 the 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.⌋ (SRS_E2E_08528)

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.⌋ (SRS_E2E_08528)

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").⌋ (SRS_E2E_08528)

11.9 Restrictions on the use of COM features

The following table lists COM features with a brief description and provides a classification of restriction of use in combination with End-to-End communication protection as described in this document.
Note: This list only covers features of the BSW module COM in combination with E2E Library and E2E Protection Wrapper. It does not address features of above layers (e.g. RTE) or use-cases where the E2E Transformer is used. The latter usually is used above the BSW module LdCom.

The restriction classes are as follows:

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

COM Feature / brief description	Classification
[SRS_Com_02078] Support of endianess conversion	supported
[SRS_Com_02086] Support of Sign-Extension for received signals	supported
[SRS_Com_02042] Initialization of unused areas/ bits of an I-PDU	supported
[SRS_Com_02083] Transmission Modes	use case dependent
[SRS_Com_02082] Two different Transmission Modes	use case dependent
[SRS_Com_02084] Signal data based selection of Transmission Mode	use case dependent
[SRS_Com_02113] Signal data based transmission modes for configured serialized data	use case dependent
[SRS_Com_02046] Configuration of signal notification	supported
[SRS_Com_02089] Timeout indication mechanism on receiver-side	supported
[SRS_Com_02088] Value substitution in case of a signal timeout	use case dependent
[SRS_Com_02080] Cancelation outstanding repetitions in case of a new send request	use case dependent
[SRS_Com_02089] two configurable options to handle signal timeouts	use case dependent
[SRS_Com_02077] Signal invalidation mechanism on sender-side	use case dependent
[SRS_Com_02079] Signal invalidation mechanism on receiver-side	use case dependent
[SRS_Com_02087] Substitution of invalid value by configurable data value	use case dependent
[SRS_Com_2088] Substitution of the last received value by the init value in case of signal timeout	use case dependent
[SRS_Com_00218] Starting/ Stopping communication of I-PDU groups	supported
[SRS_Com_00192] Enabling/ disabling reception deadline monitoring of I-PDU groups	use case dependent
[SRS_Com_02041] Consistent transfer of complex data types	supported
[SRS_Com_02091] Placement of large or dynamical length signals	not supported
[SRS_Com_02092] Support only one dynamic length signal per I-PDU	not supported
[SRS_Com_02093] Dynamic length signal must be placed last in I PDU	not supported
[SRS_Com_02094] Dynamic length signals must be of type UINT8[n]	not supported
[SRS_Com_02095] TP shall be used to fragment and reassemble large signals and dynamical signals	not supported
[SRS_Com_02030] Identify if a signal/signal group is updated by the sender	use case dependent
[SRS_Com_02058] Deadline monitoring of receiving updated signals/signal groups	use case dependent
[SRS_Com_02099] I-PDU Counter mechanism	use case dependent
[SRS_Com_02100] I-PDU Counter configuration	use case dependent
[SRS_Com_02101] Transmission and reception using I-PDU Counter	use case dependent
[SRS_Com_02102] I-PDU Counter error handling	use case dependent
[SRS_Com_02103] I-PDU Replication mechanism	use case dependent
[SRS_Com_02104] I-PDU replication configuration	use case dependent
[SRS_Com_02105] Transmission and reception using I-PDU Replication	use case dependent
[SRS_Com_02106] I-PDU Replication error handling	use case dependent
Minimum Delay Time	use case dependent

COM Feature / brief description	Classification
Filtering at receiver side (e.g. COM273)	use case dependent
Filtering at sender side	use case dependent
Multiple Signal groups within an I-PDU	use case dependent

Table 11-3: Classification of COM features

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

Note: this has been moved from chapter 12.

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 priciples 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.

11.10.1 Basic principles

Typical basic priciples 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-determinded (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.

Note: the transition to "Startup" involves proper initialization of the E2EPW, either by calling E2EPW_ReadInit_<p>_<o> or by ensuring that the data structures were initialized by the startup code. By choosing initial values for received data elements that result in a CRC error, the state machine remains in state "Startup" and the E2EPW is reset until valid data is received. Then, E2EPW will return E2EPW_STATUS_INITIAL and the state machine changes its state to "Initialized".

11.10.2 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-Libray)
- 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 11-3:

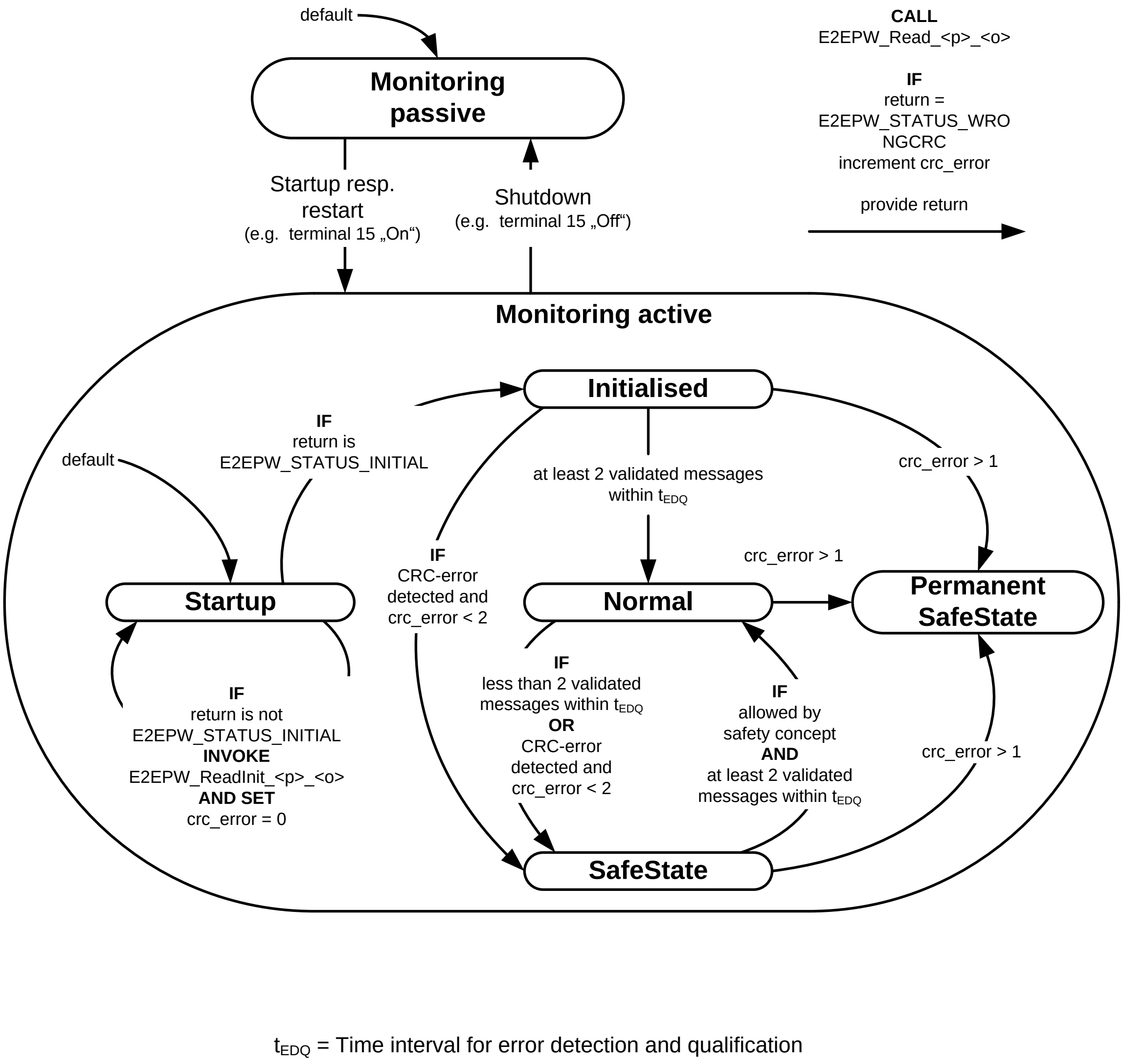


Figure 11-3: Example for a monitoring function to determine the integrity of communication within a receiver

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} = 160ms and a regular cycle time of 20ms the value N = 160ms/20ms = 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 invalid. Depending on the systems functional needs or it 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 occured).
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. transistion into a permanent invalid if the second CRC-error is detected), the probability that more than one undetected errouneous 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.

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 in different ways (each explained in a separate section of this chapter):

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

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

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

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

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

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

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

All necessary options, enabling to generate the code for the described solutions are available in AUTOSAR configuration, defined in System Template [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 ISignalIPdu metaclass (representing the I-PDU).

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

[UC_E2E_00271]⌈ A given I-PDU, if protected by E2E, shall be protected by only one E2E mechanism.⌋ (SRS_E2E_08528)

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-Cs[1], 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.

[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.

[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.
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].⌋ (SRS_E2E_08528)

[UC_E2E_00249]⌈ The integrity of the operation of E2E Protection Wrapper (for transmitting/converting safety-related data) shall be guaranteed.⌋ (SRS_E2E_08528)

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

[UC_E2E_00288]⌈ Each E2E Protection Wrapper function shall not be called concurrently.⌋ (SRS_E2E_08528)

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

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

12.1.1 Functional overview

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

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

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

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

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

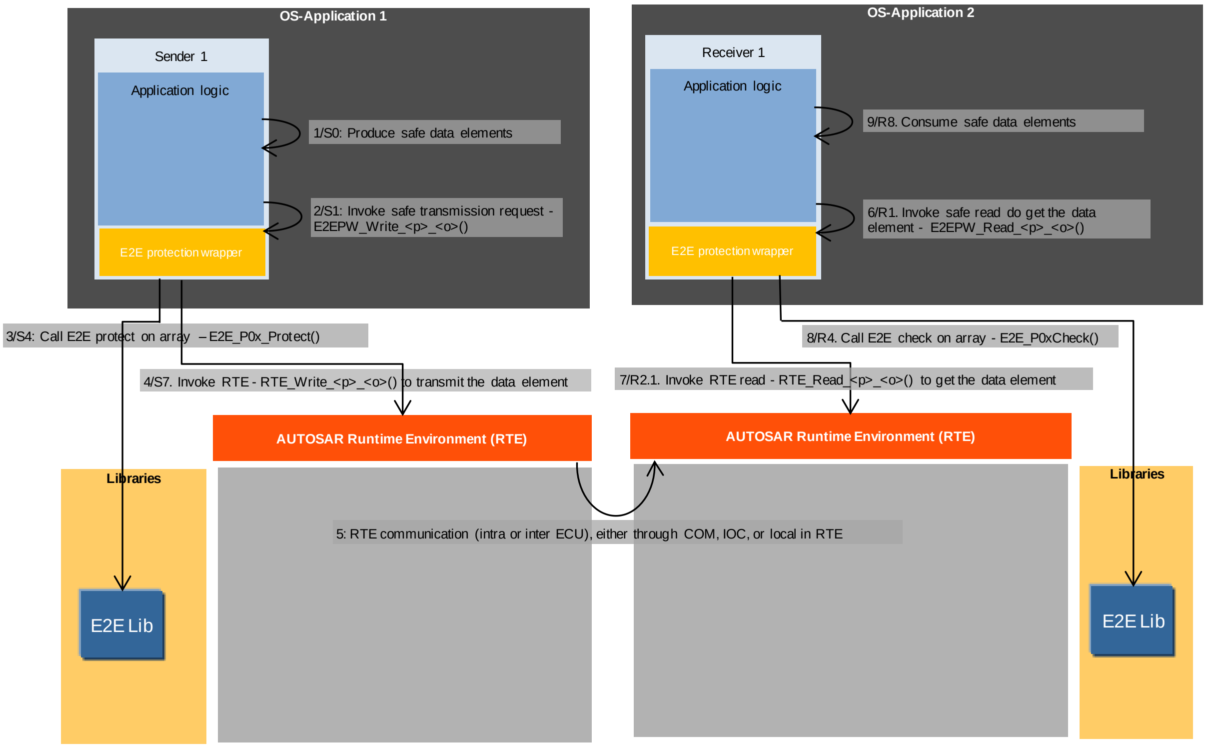


Figure 12-1: Example E2E Protection Wrapper - overall flow

12.1.2 Application scenario with Transmission Manager

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

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

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

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

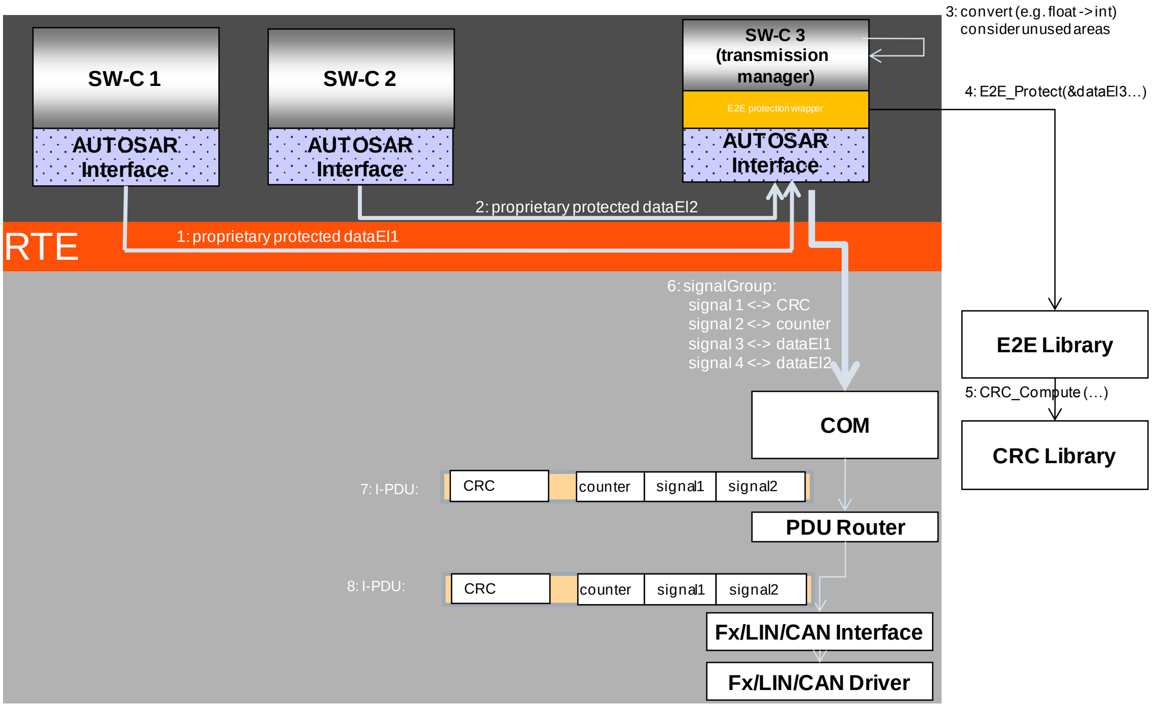


Figure 12-2: Example Transmission Manager – sender ECU

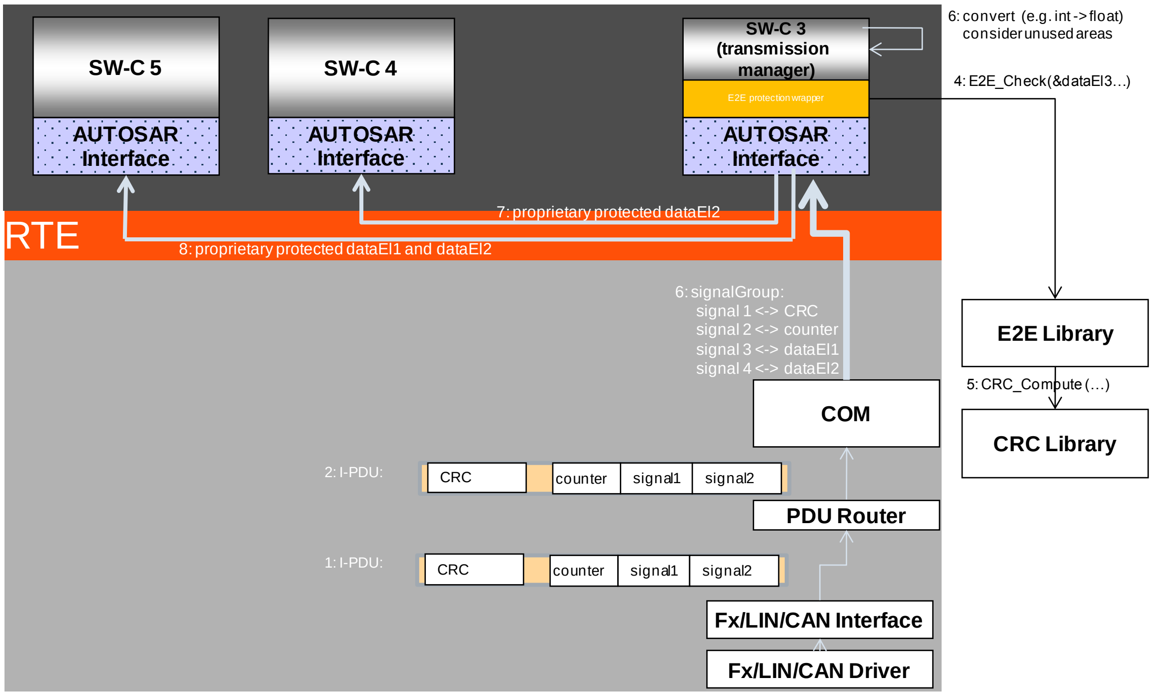


Figure 12-3: Example Transmission Manager – receiver ECU

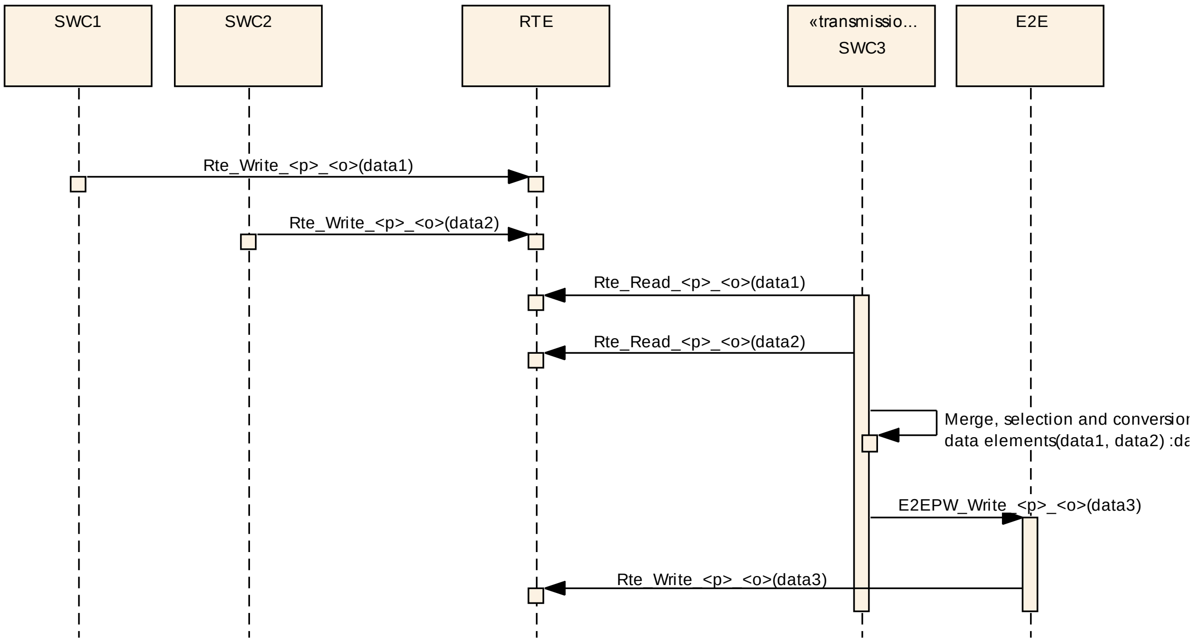


Figure 12-4: Example Transmission Manager – sender ECU sequence

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

[UC_E2E_00213]⌈ The implementation of the Transmission Manager (as a safety-related Software Component), shall comply with the requirements for the development of safety-related software for automotive domain.⌋ (SRS_E2E_08528)

12.1.3 Application scenario with E2E Manager and Conversion Manager

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

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

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

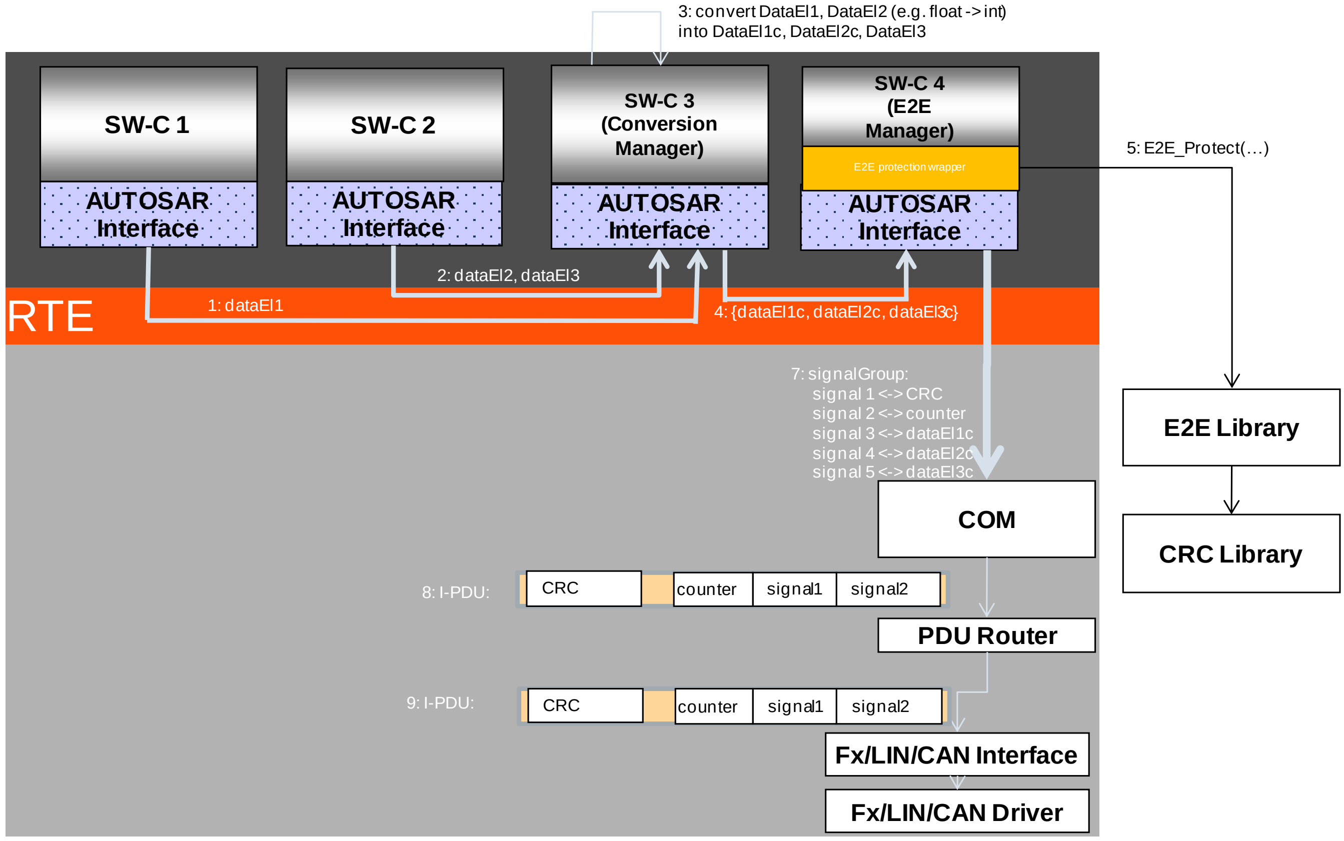


Figure 12-5: E2E Manager and Conversion Manager – sender ECU

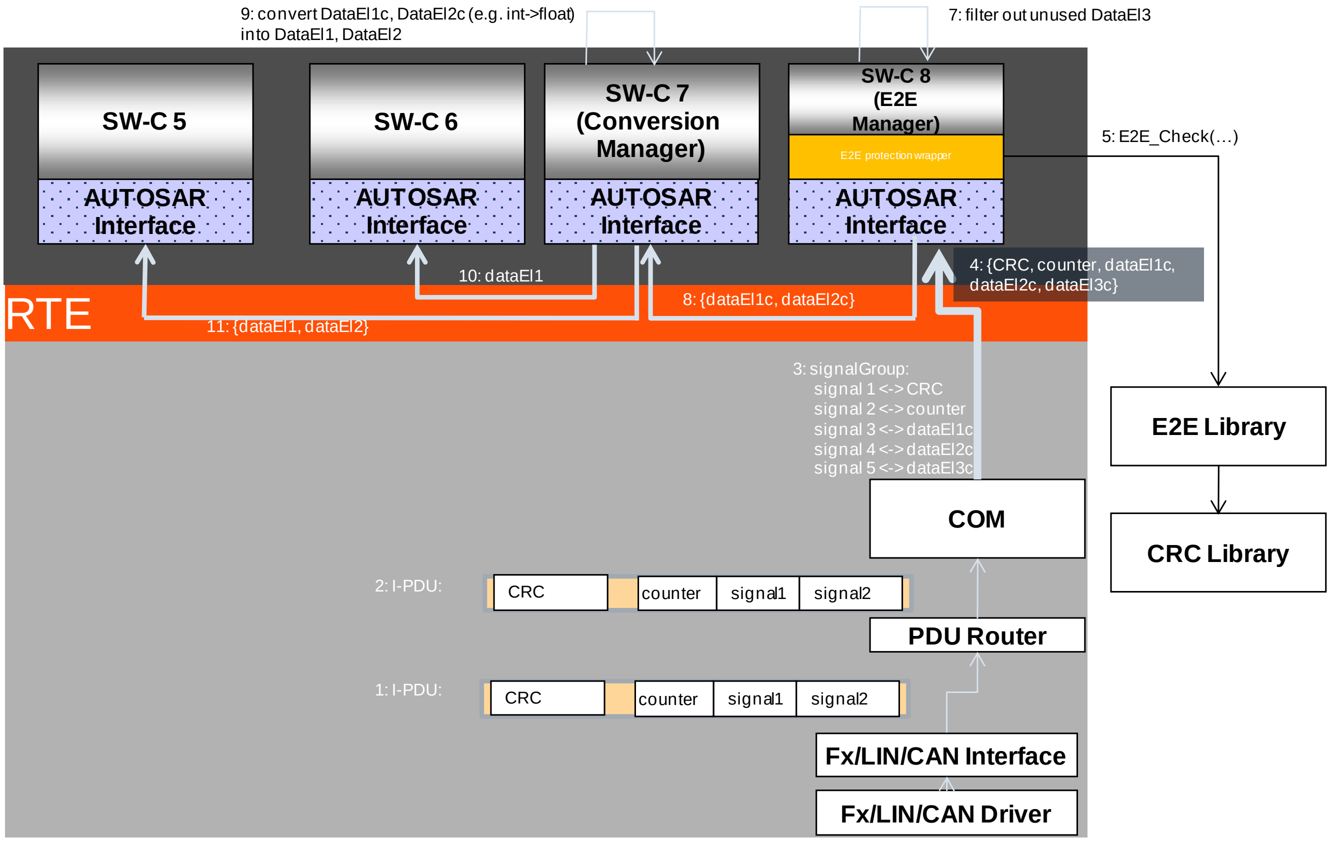


Figure 12-6: E2E Manager and Conversion Manager – receiver ECU

In the above example, the SW-Cs of sender ECU generate three data elements (dataEl1, dataEl2 and dataEl3) but the SW-Cs of receiver ECU use only two data elements (dataEl1 and dataEl2). The unused DataEl3c is not delivered to Conversion Manager. Thanks to this, if due to e.g. system evolution, the definition of DataEl3 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, dataEl3c) are not provided:

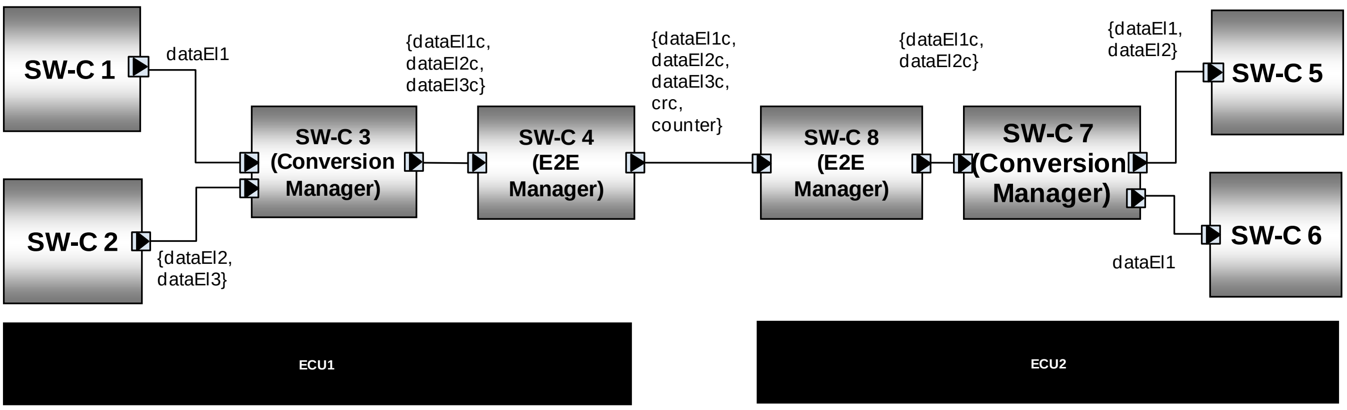


Figure 12-7: E2E Manager and Conversion Manager - system configuration

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

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

```
/* the input complex data element contains primitive data elements
   unused by other SW-Cs of the ECU */
typedef struct {
      uint8 crc;
      uint8 counter;
      uint16 dataEl1c;
      uint16 dataEl2c;
      uint16 dataEl3c;
} 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 dataEl1c;
      uint16 dataEl2c;
} Outputswc8DataType;

Inputswc8Type Inputswc8;
Outputswc8Type Outputswc8;
```

```
...

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

Outputswc8Type.dataEl1c = Inputswc8Type.dataEl1c;
Outputswc8Type.dataEl2c = Inputswc8Type.dataEl2c;
```

[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⌋ (SRS_E2E_08528)

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

[UC_E2E_00275]⌈ The output data element shall contain the subset of primitive data elements of those of the corresponding input data element (in particular, they may be equal).⌋ (SRS_E2E_08528)

In the example above, Outputswc8 contains the subset of attributes of Inputswc8. It does not contain dataEl3c, 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 dataEl1c and dataEl2c 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. ⌋ (SRS_E2E_08528)

[UC_E2E_00273]⌈ The E2E Manager SW-C at receiver ECU shall filter out the data elements that are not used by the SW-Cs of the ECU. The E2E Manager SW-C at receiver ECU shall forward to Conversion Manager SW-C only the data elements that are used by Conversion Manager SW-C.⌋ (SRS_E2E_08528)

12.1.4 File structure

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

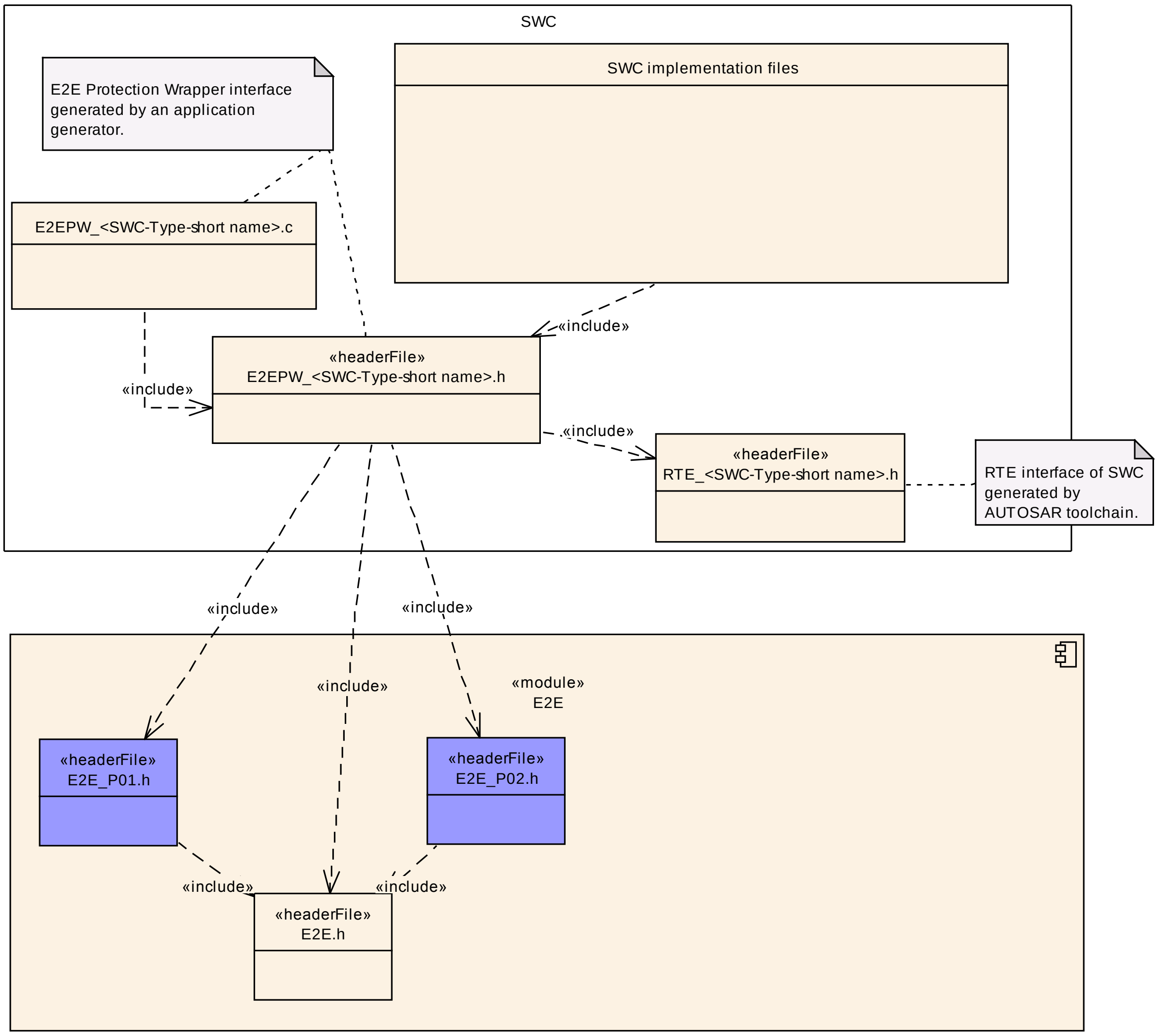


Figure 12-8: E2E File dependencies

[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.⌋ (SRS_E2E_08528)

[UC_E2E_00240]⌈ E2EPW_<SWC-Type-short name>.c shall include E2EPW_<SWC-Type-short name>.h.⌋ (SRS_E2E_08528)

[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.⌋ (SRS_E2E_08528)

[UC_E2E_00242]⌈ The SW-C implementation files that invoke E2E Protection Wrapper functions shall include E2EPW_<SWC-Type-short name>.h⌋ (SRS_E2E_08528)

[UC_E2E_00256]⌈ The E2E Protection Wrapper shall ensure the integrity of the safety-related data elements.⌋ (SRS_E2E_08528)

[UC_E2E_00257]⌈ The implementation of the E2E Protection Wrapper (as a safety-related Software Component) shall comply with the requirements for the development of safety-related software for the automotive domain. ⌋ (SRS_E2E_08528)

12.1.5 Methodology

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

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

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

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

[UC_E2E_00248]⌈ The E2E Protection Wrapper shall be generated for the complex data elements (represented by VariableDataPrototype metaclass) for which the corresponding EndToEnd* metaclasses are defined.⌋ (SRS_E2E_08528)

[UC_E2E_00289]⌈ If the E2EProtection is done in the E2E Wrapper then both EndToEndProtectionISignalIPdu and EndToEndProtectionVariablePrototype shall be defined.⌋ (SRS_E2E_08528)

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

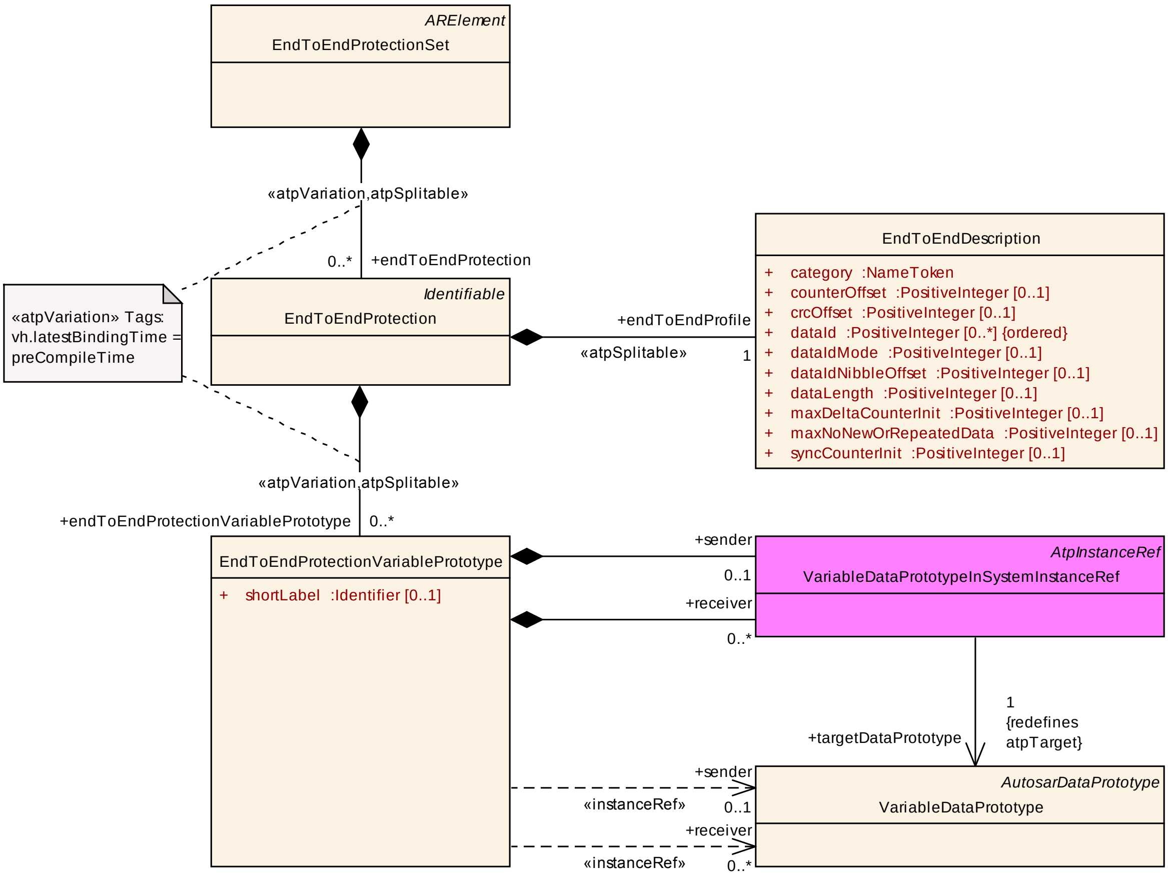


Figure 12-9: Release R4.0.1 and newer: E2E Protection Wrapper configuration (hardcopy from DOC_EndtoEndProtection)

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

Apart from configuring EndToEndProtectionVariablePrototype, further settings involve the mapping signal groups to I-PDUs, which is done according to System Template [12]:

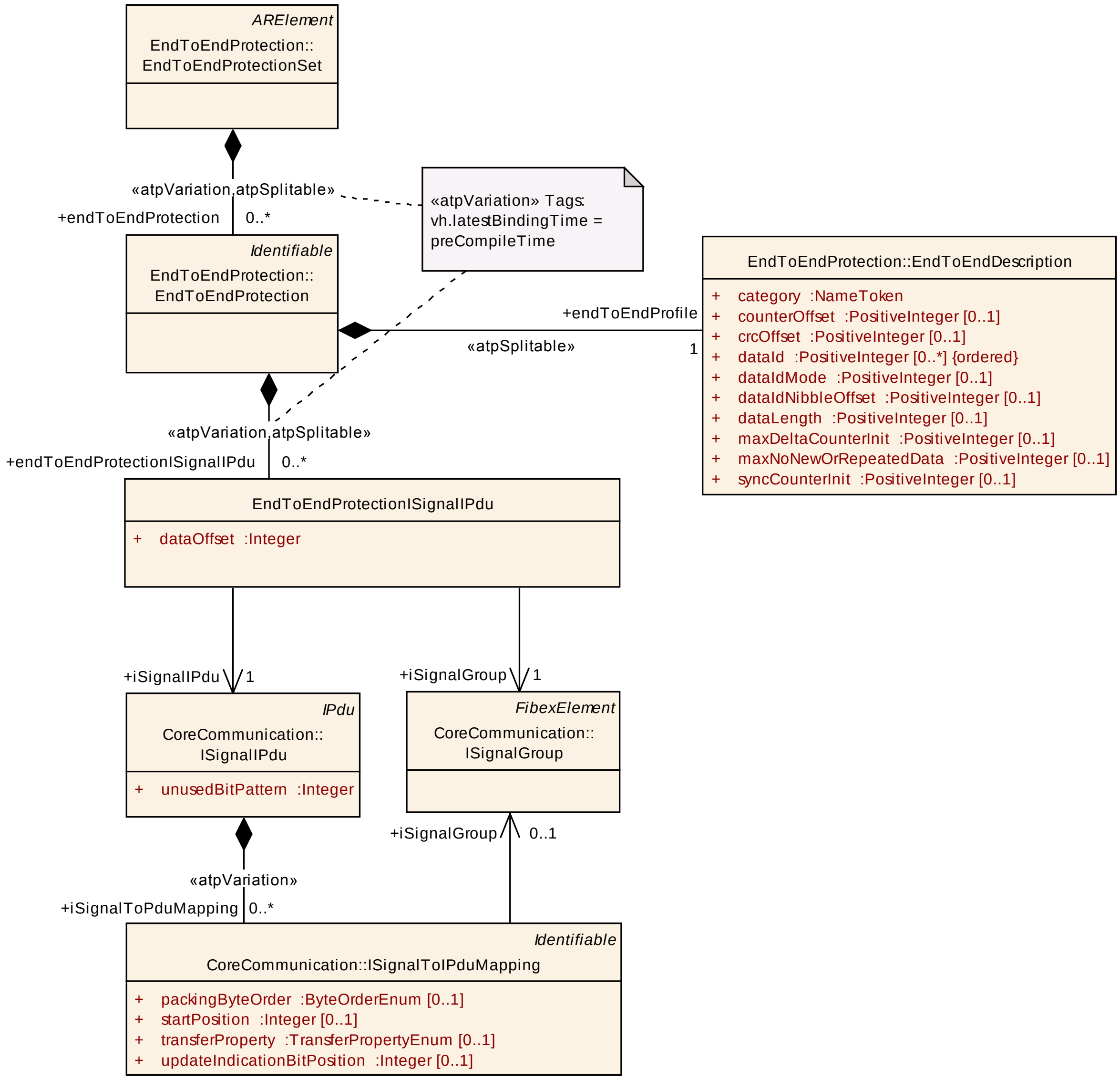


Figure 12-10: Release R4.0.1 and newer: E2E Protection Wrapper configuration (hardcopy from DOC_PduEndToEndProtection)

The important settings are:

1. ISignalIPdu (represents an I-PDU)
    a. ISignalIPdu.unusedBitPattern:bits that are not used in an I-PDU,

2. ISignalToIPduMapping: describes the mapping of signals to I-PDUs,
    a. ISignalToIPduMapping.startPosition: offset in bits of a signal in the I-PDU,

3. EndToEndProtectionISignalIPdu: association of one E2E protection to a one I-PDU and to one signal group,
    a. EndToEndProtectionISignalIPdu.dataOffset: offset in bits of the signal group in the I-PDU.

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

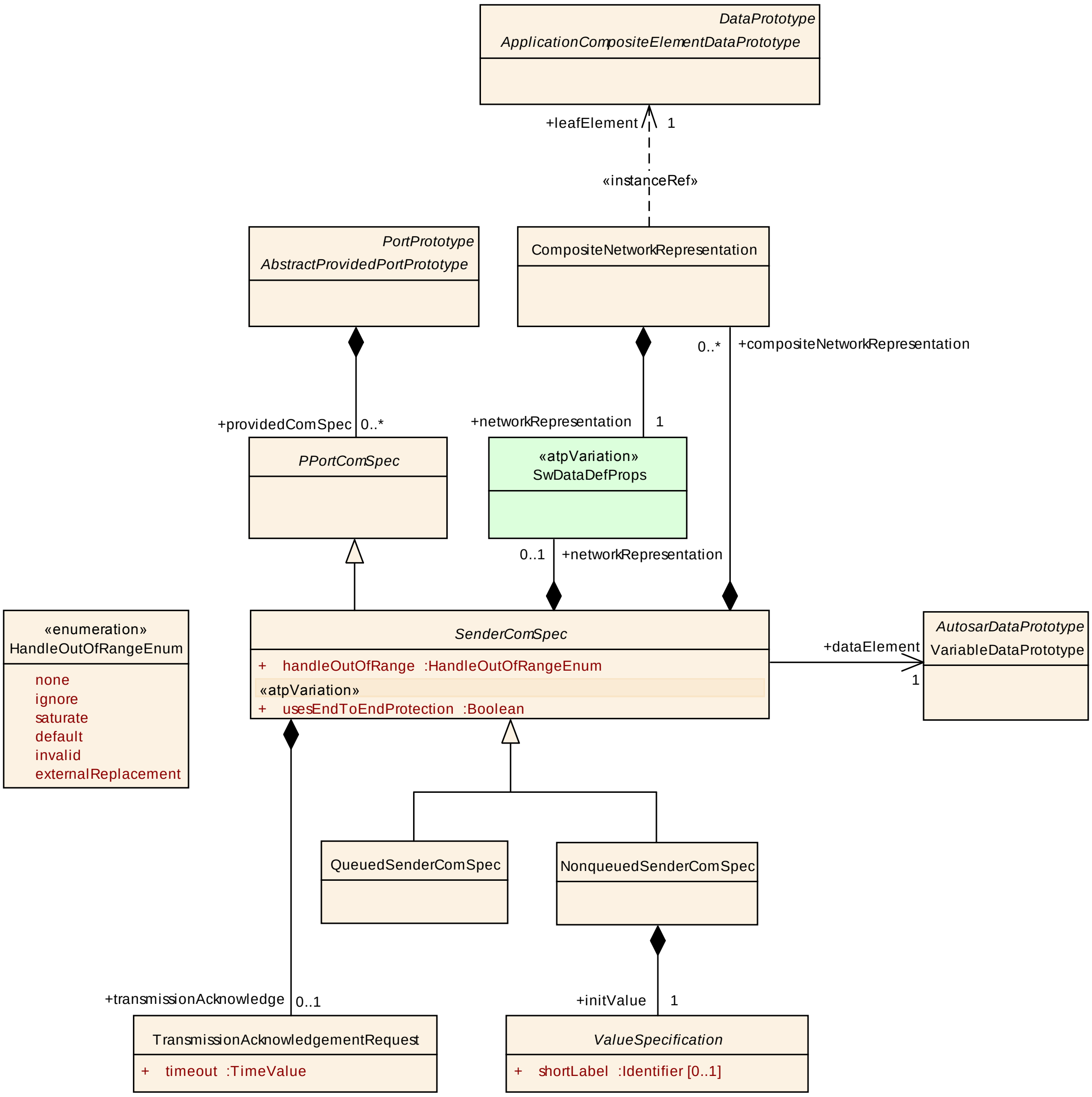


Figure 12-11: Release R4.0.1 and newer: SenderComSpec (hardcopy from DOC_SenderComSpec)

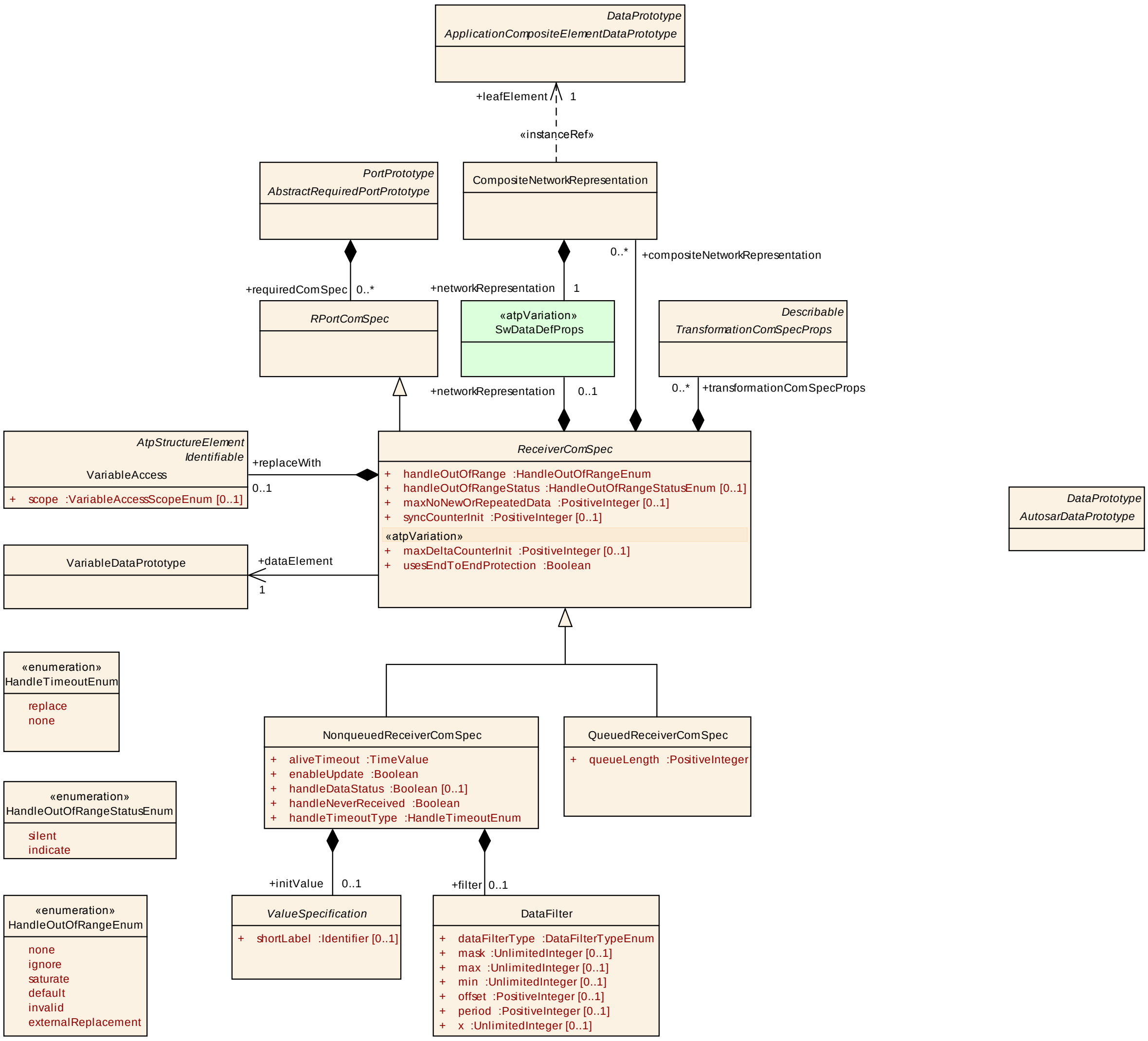


Figure 12-12: Release R4.0.1 and newer: ReceiverComSpec (hardcopy from DOC_ReceiverComSpec)

12.1.6 Error classification

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

[UC_E2E_0302]:
Where applicable, the following error status shall be used by E2E Wrapper functions within byte 3 of the return value, in addition to the error codes already defined by [SWS_E2E_00047] (chapter 7.2.1):

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	Production	`E2EPW_STATUS_NONEWDATA`	0x1

consequently executed.			
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_WRONGCRC	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 < 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.	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 (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.	Production	E2EPW_STATUS_WRONGSEQUENCE	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/stati defined in UC_E2E_0302 and UC_E2E_0303 according to the column “How do caller of E2E shall handle it”.⌋ (SRS_E2E_08528)

In other words, the E2E libary 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 Proteciton wrapper, this means that the optional parameter <instance> shall never be present.⌋ (SRS_E2E_08528)

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

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

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

- For byte 1, 2 and 3 the set of return values ranging from 0x00 to 0x7F (i.e. decimal 0 to 127) is restricted for usage within AUTOSAR specifications only and shall not be used for proprietary return values that are not part of AUTOSAR specifications.
- For byte 1, 2 and 3 the set of return values ranging from 0x80 to 0xFE (i.e. decimal 128 to 254) is not restricted and shall be used for proprietary implementation specific return values that are not part of AUTOSAR specifications.
- For byte 1, 2 and 3 the value 0xFF (i.e. decimal 255) represents the invalid value. .⌋ (SRS_E2E_08527)

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

[UC_E2E_00328]⌈ Redundant wrapper routines shall use separate configuration and state data structures for each of the redundant channels.⌋ (SRS_E2E_08527)
E.g. use config1_<p>_<o>/state1_<p>_<o> for channel 1 and config2_<p>_<o>/state2_<p>_<o> for channel 2, as indicated in the code example in 12.1.9.1.

12.1.7.1 Single channel wrapper routines and init routines

12.1.7.1.1 E2EPW_Write_<p>_<o>

[UC_E2E_00279]⌈

Service name:	E2EPW_Write_<p>_<o>	
Syntax:	`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	The byte 0 (lowest byte) is the status of Rte_Write function: RTE_E_COM_STOPPED - the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only) RTE_E_SEG_FAULT - a segmentation violation is detected in the handed over parameters to the RTE API. No transmission is executed RTE_E_OK - data passed to communication service successfully The byte 1 is the status of runtime 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 The byte 2 is the return value of E2E_PXXProtect function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function E2E_PXXProtect completed successfully The byte 3 is a placeholder for future use and takes the following values E2E_E_OK - default case
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.	

⌋ (SRS_E2E_08528)

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

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
2. Invoke E2E Library function E2E_PXXProtect()
3. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array , store the computed CRC/Counter in the data element
4. Invoke Rte_Write_<p>_<o>()⌋ (SRS_E2E_08528)

See also Figure 12-13: E2EPW_Write sequence diagram and Figure 12-18: E2EPW_Write activity diagram.

12.1.7.1.2 E2EPW_WriteInit_<p>_<o>

[UC_E2E_00300]⌈

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

⌋ (SRS_E2E_08528)

[UC_E2E_00301]⌈ The function E2EPW_WriteInit_<p>_<o> shall initialize the E2E_PXXProtectStateType_<p>_<o> with the following values:

Counter = 0⌋ (SRS_E2E_08528)

12.1.7.1.3 E2EPW_Read_<p>_<o>

[UC_E2E_00165]⌈

Service name:	E2EPW_Read_<p>_<o>
Syntax:	`uint32 E2EPW_Read_<p>_<o>( Rte_Instance <instance>, -- <data> )`
Service ID[hex]:	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, 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 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 The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. 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 or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults. 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 behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
Description:	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	

⌋ (SRS_E2E_08528)

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

1. Invoke Rte_Read_<p>_<o>()
2. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
3. Invoke E2E Library function E2E_PXXCheck()
4. Do the deserialization check.⌋ (SRS_E2E_08528)

See also Figure 12-14: E2EPW_Read sequence diagram and Figure 12-15: E2EPW_Read activity diagram.

12.1.7.1.4 E2EPW_ReadInit_<p>_<o>

[UC_E2E_00296]⌈

Service name:	E2EPW_ReadInit_<p>_<o>
Syntax:	`Std_ReturnType E2EPW_ReadInit_<p>_<o>( Rte_Instance <instance> )`
Service ID[hex]:	0x16
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_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.	

⌋ (SRS_E2E_08528)

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

LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
NoNewOrRepeatedDataCounter = 0
SyncCounter = 0⌋ (SRS_E2E_08528)

12.1.7.2 Redundant wrapper routines

12.1.7.2.1 E2EPW_Write1_<p>_<o>

[UC_E2E_00261]⌈

Service name:	E2EPW_Write1_<p>_<o>	
Syntax:	`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	The byte 0 (lowest byte) is equal to E2E_E_OK (because Rte_Write 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_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 eror 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 The byte 2 is the return value of E2E_PXXProtect function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2E_E_OK - Function E2E_PXXProtect completed successfully The byte 3 is a placeholder for future use and takes the following values: E2E_E_OK - default case
Description:	It protects data with E2E Library function E2E_PXXProtect. it does not call the corresponding RTE_Write function.	

⌋ (SRS_E2E_08528)

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

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
2. Invoke E2E Library function E2E_PXXProtect()
3. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, store the computed CRC/Counter in the data element.⌋ (SRS_E2E_08528)

See also Figure 12-19: E2EPW_Write1 activity diagram.

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

⌋ (SRS_E2E_08528)

[UC_E2E_00264]⌈ The function E2EPW_Write2_<p>_<o>() shall:
1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
2. Invoke E2E Library function E2E_PXXProtect()
3. Execute voting on control fields between Write1 and Write2

4. Invoke Rte_Write_<p>_<o>() .⌋ (SRS_E2E_08528)

See also Figure 12-20: E2EPW_Write2 activity diagram.

12.1.7.2.3 E2EPW_WriteInit1_<p>_<o>

[SWS_E2E_00318]⌈

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

⌋ (SRS_E2E_08528)

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

Counter = 0.⌋ (SRS_E2E_08528)

12.1.7.2.4 E2EPW_WriteInit2_<p>_<o>

[SWS_E2E_00319]⌈

Service name:	E2EPW_WriteInit2_<p>_<o>	
Syntax:	`uint8 E2EPW_WriteInit2_<p>_<o>( Rte_Instance <instance> )`	
Service ID[hex]:	0x18	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	The byte 0 is the status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or

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

⌋ (SRS_E2E_08528)

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

Counter = 0.⌋ (SRS_E2E_08528)

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 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 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 The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. 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 or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults. 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 behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
Description:	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	

⌋ (SRS_E2E_08528)

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

1. Invoke Rte_Read_<p>_<o>()
2. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
3. Invoke E2E Library function E2E_PXXCheck()
4. Do the deserialization check.⌋ (SRS_E2E_08528)

See also Figure 12-16: E2EPW_Read1 activity diagram.

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.
	<data>	The received data to be checked. The parameter is IN, because this function re-performs the checks on the already received data (by E2EPW_Read1_<p>_<o>). Nothing is changed in <data>. The pointer to the IN parameter <data> must remain valid until the function call returns.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint32	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

		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 The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. 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 or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults. 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 behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
Description:	The function re-checks the data received with corresponding function Read1 by means of execution of E2E_PXXCheck.	

⌋ (SRS_E2E_08528)

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

1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
2. Invoke E2E Library function E2E_PXXCheck()
3. Do the deserialization check.⌋ (SRS_E2E_08528)

See also Figure 12-17: E2EPW_Read2 activity diagram.

12.1.7.2.7 E2EPW_ReadInit1_<p>_<o>

[SWS_E2E_00320]⌈

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

⌋ (SRS_E2E_08528)

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

LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
NoNewOrRepeatedDataCounter = 0
SyncCounter = 0.⌋ (SRS_E2E_08528)

12.1.7.2.8 E2EPW_ReadInit2_<p>_<o>

[SWS_E2E_00321]⌈

Service name:	E2EPW_ReadInit2_<p>_<o>
Syntax:	`uint8 E2EPW_ReadInit2_<p>_<o>(`

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

⌋ (SRS_E2E_08528)

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

LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
NoNewOrRepeatedDataCounter = 0
SyncCounter = 0.⌋ (SRS_E2E_08528)

12.1.8 E2EPW Routines Diagrams

12.1.8.1 Sequence Diagrams – Read and Write

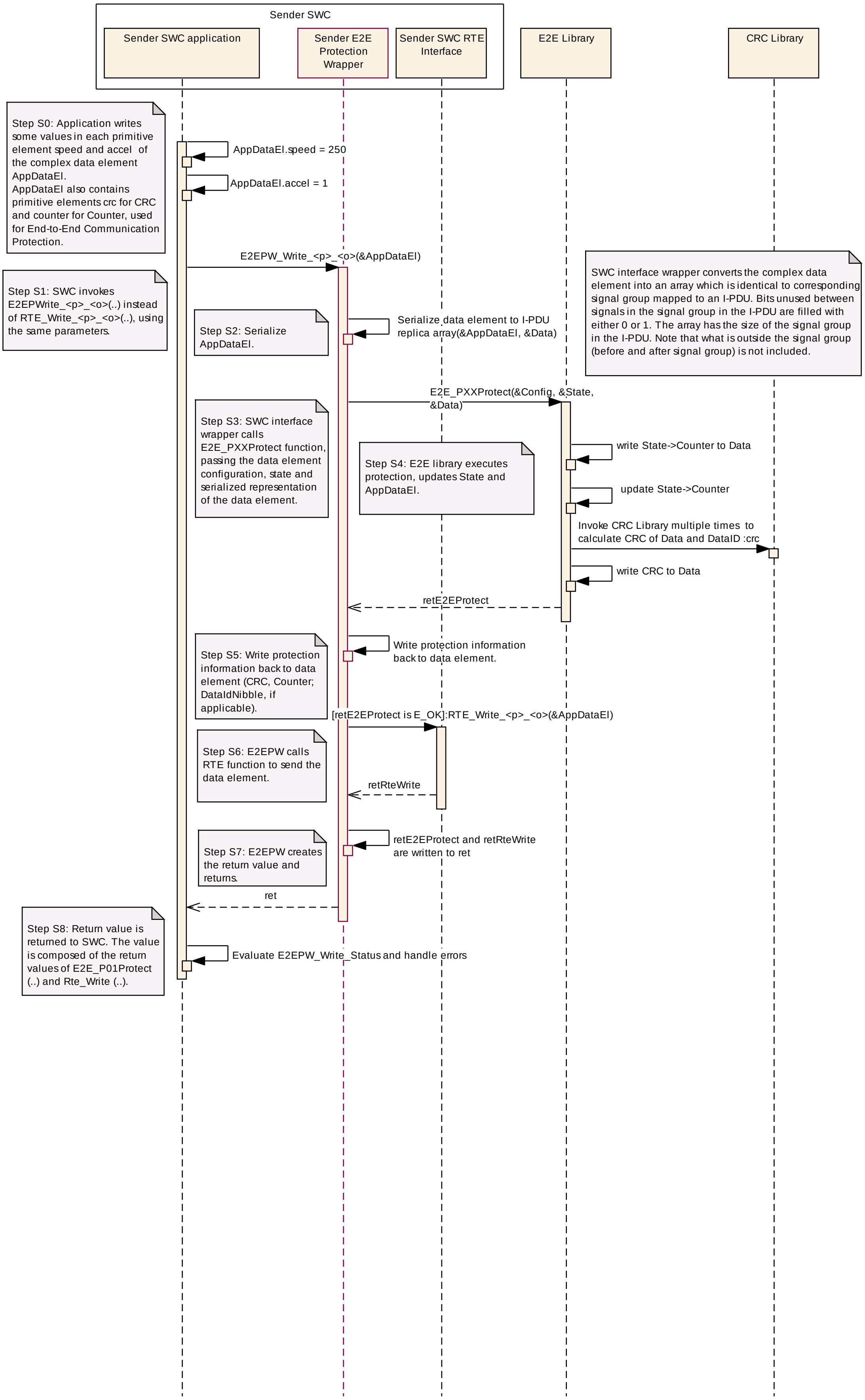


Figure 12-13: E2EPW_Write sequence diagram

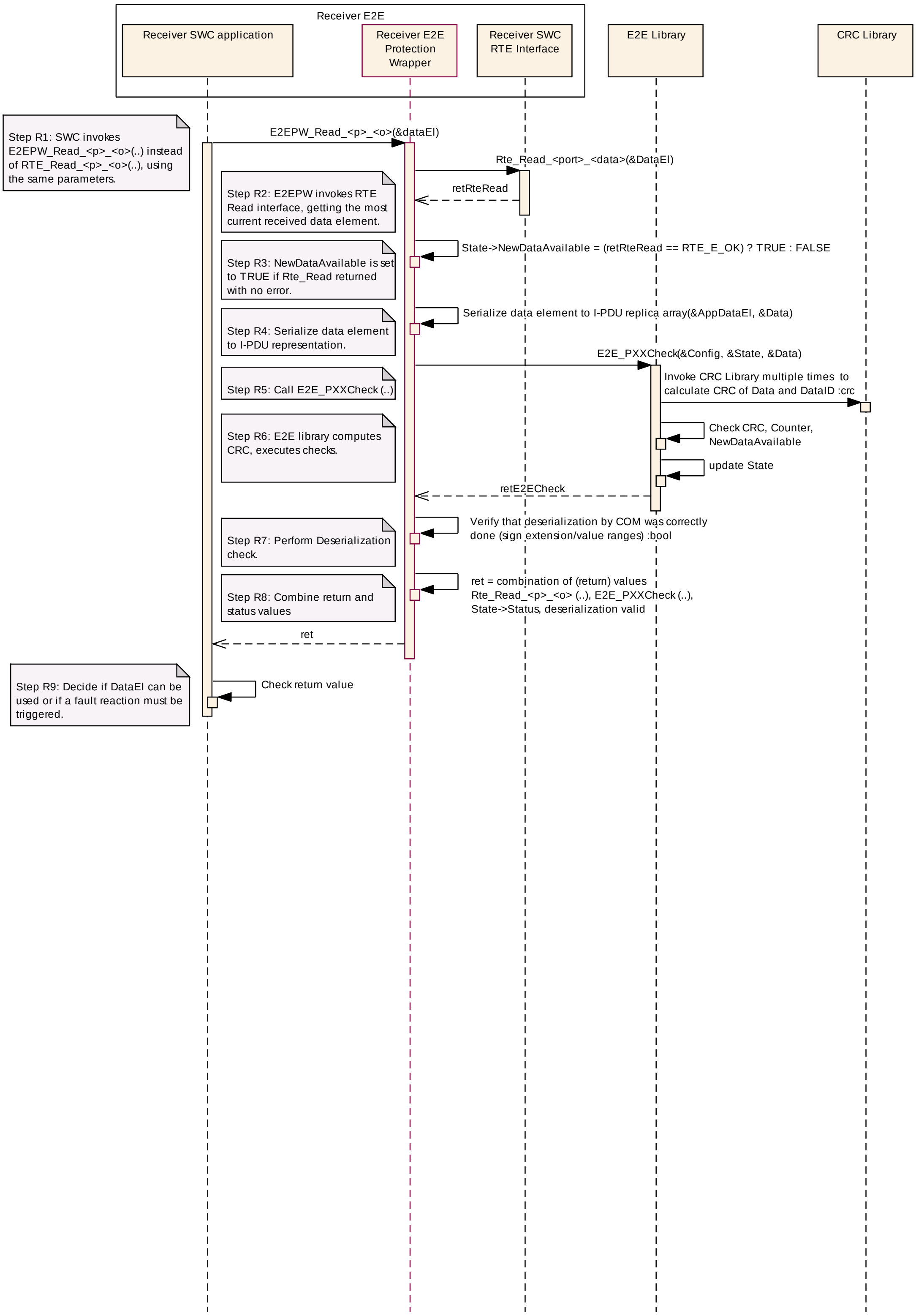


Figure 12-14: E2EPW_Read sequence diagram

12.1.8.2 Activity Diagrams – E2EPW Read, Read1 and Read2

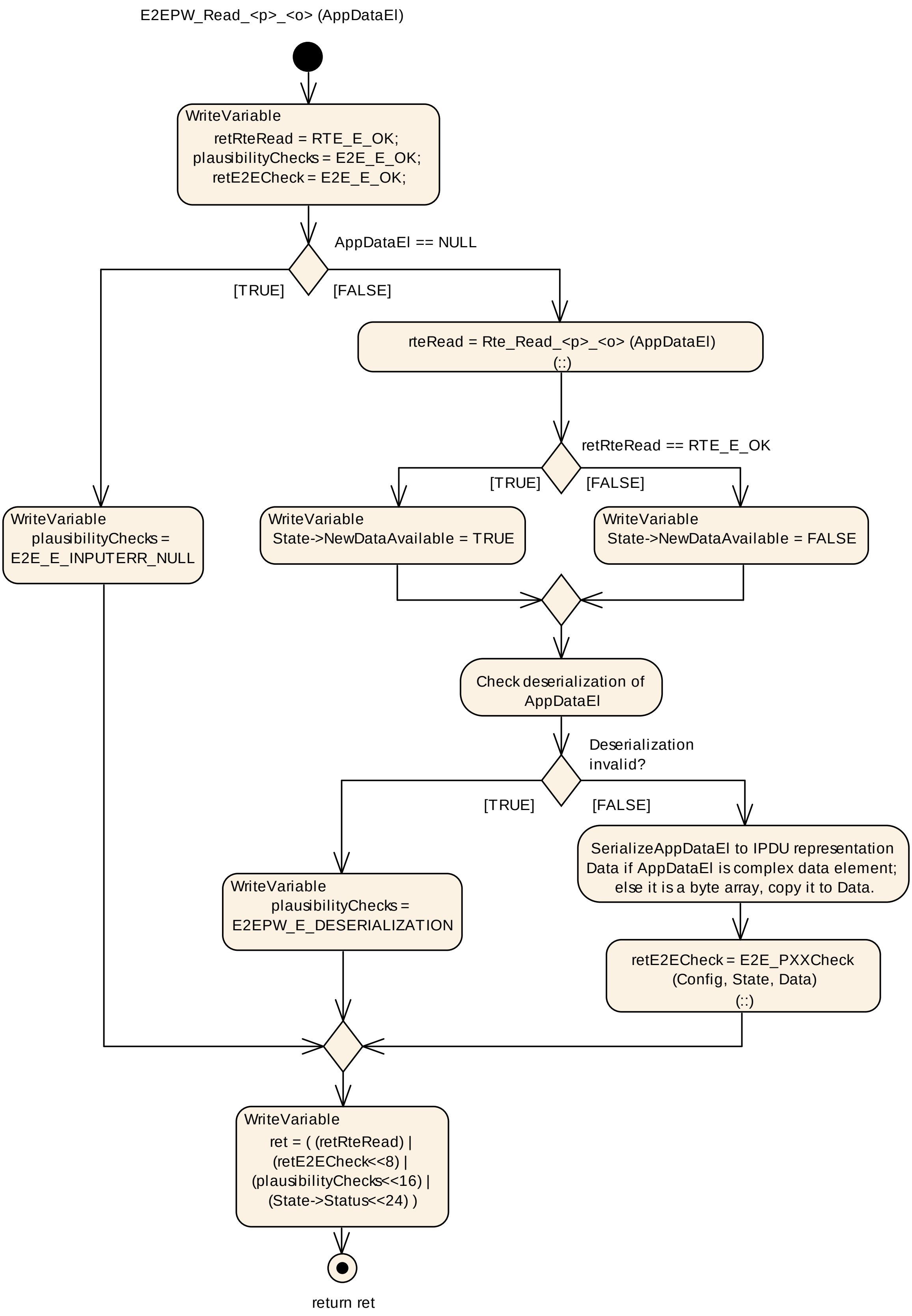


Figure 12-15: E2EPW_Read activity diagram

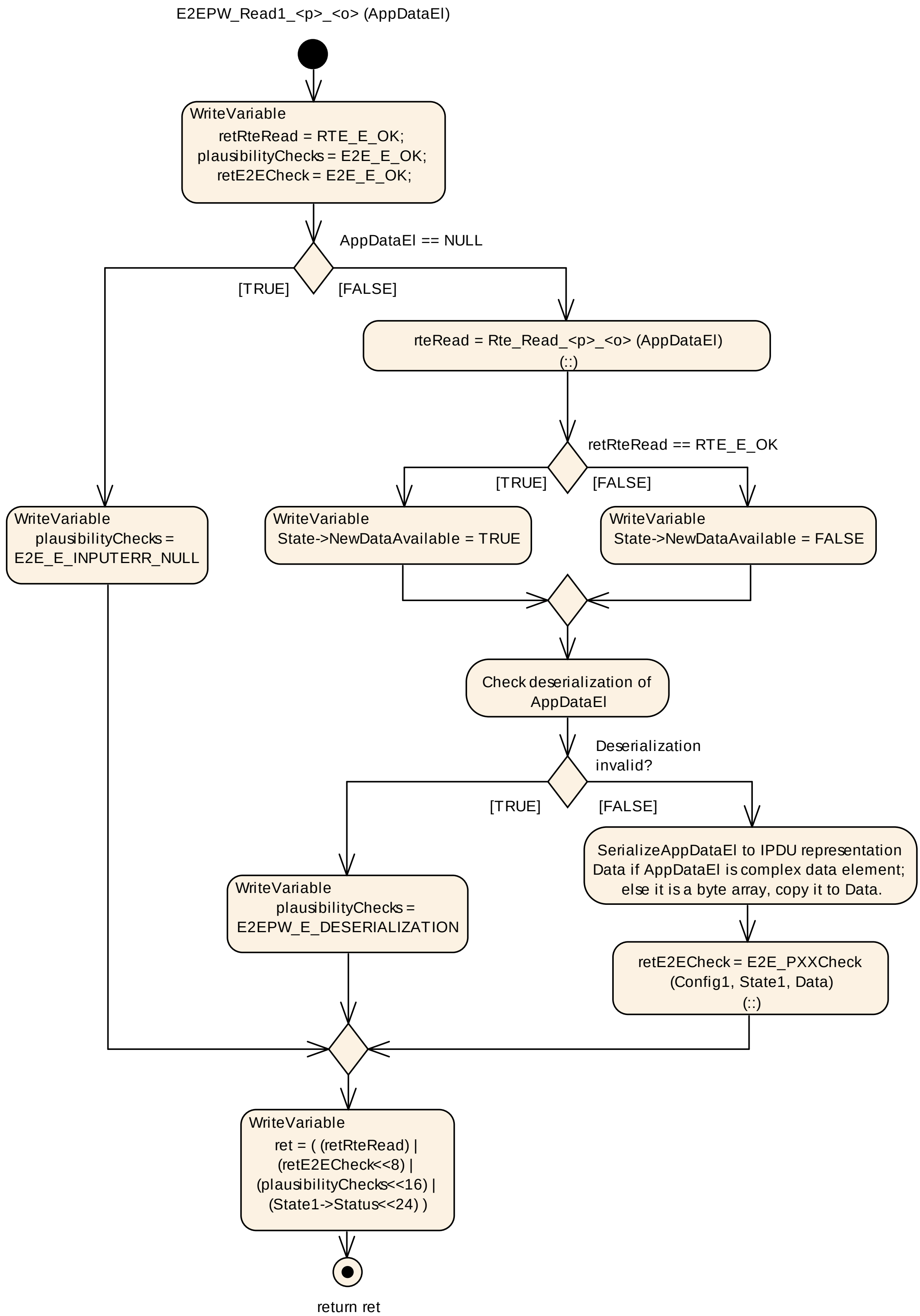


Figure 12-16: E2EPW_Read1 activity diagram

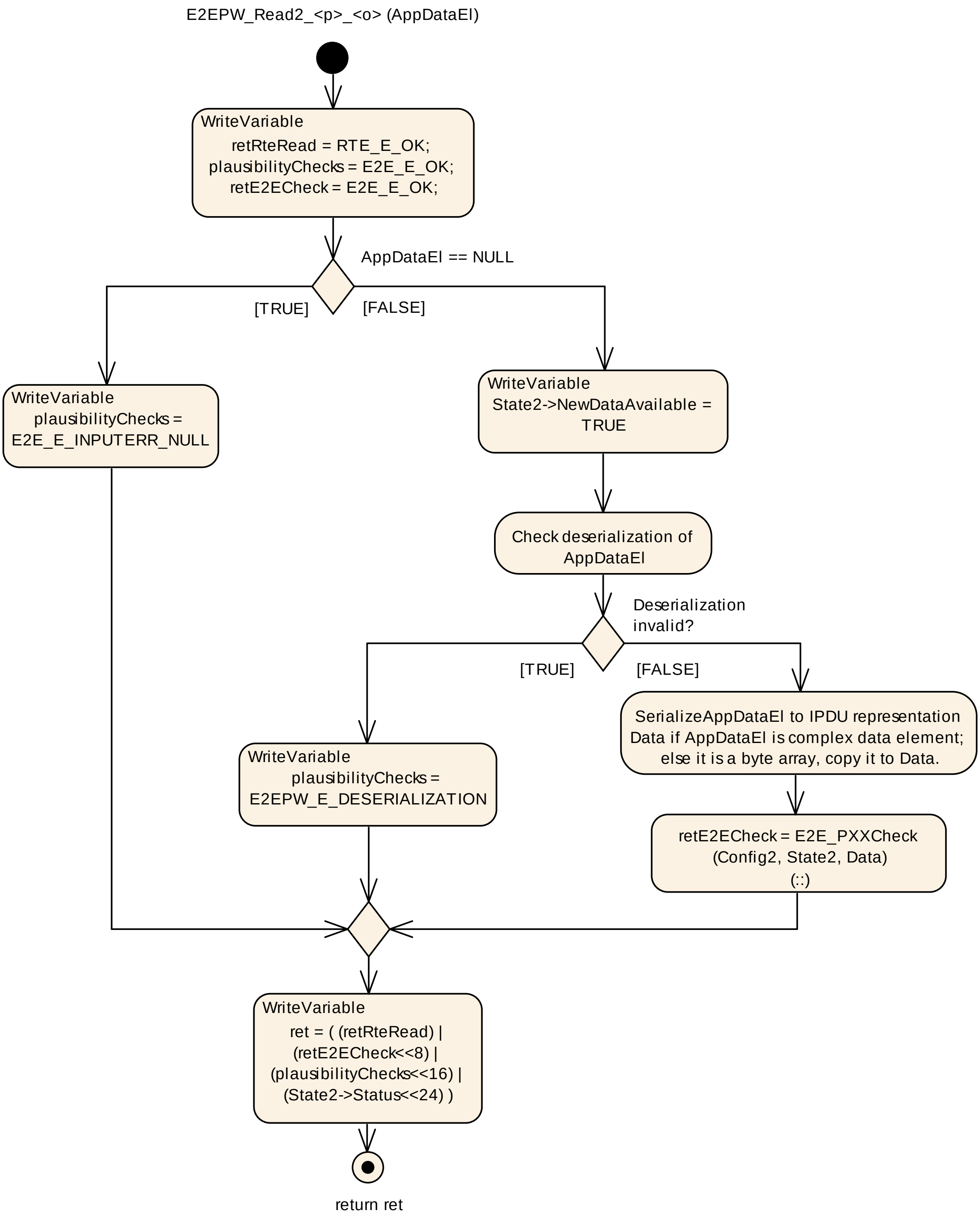


Figure 12-17: E2EPW_Read2 activity diagram

12.1.8.3 Activity Diagrams – E2EPW Write, Write1 and Write2

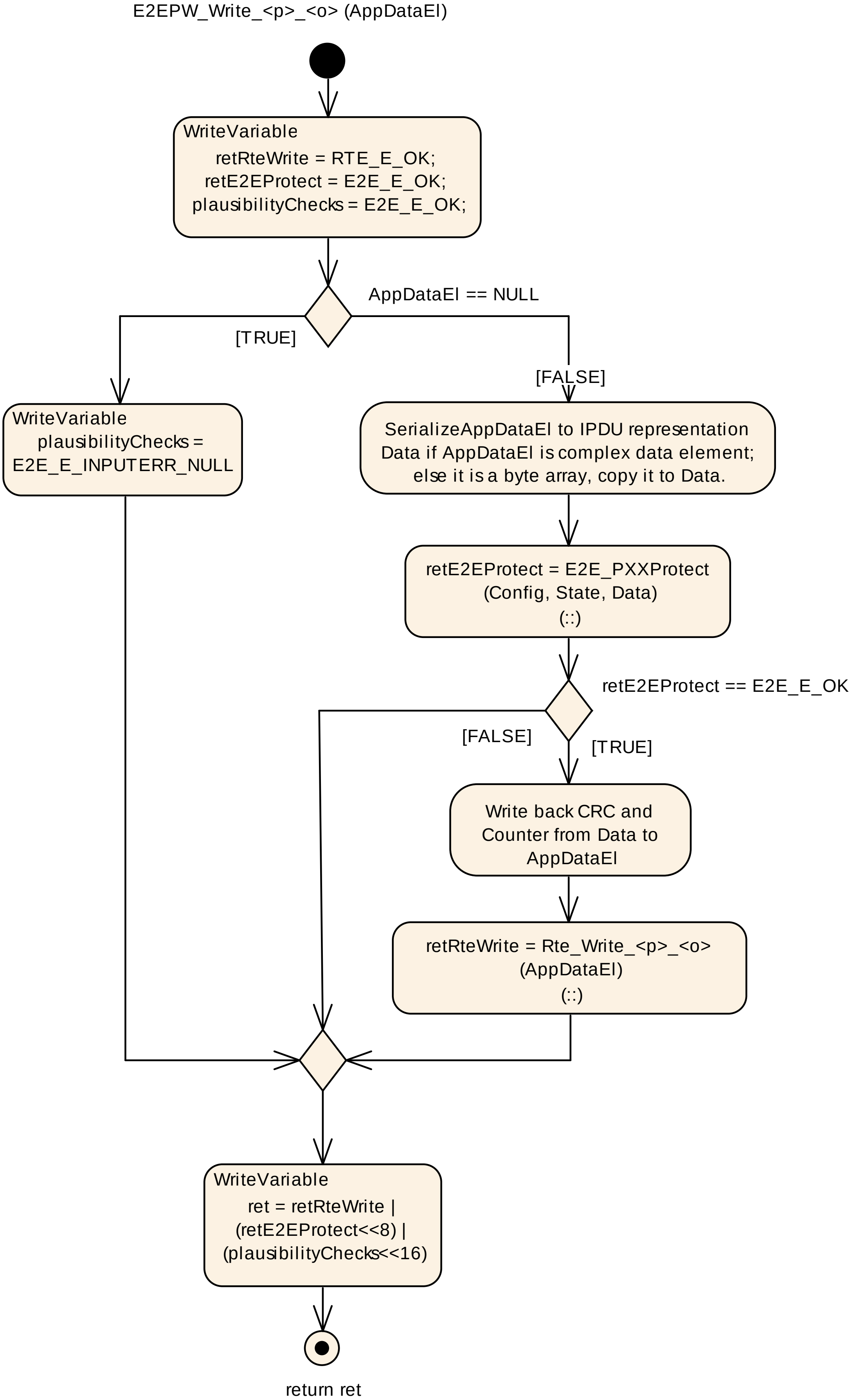

E2EPW_Write_<p>_<o> (AppDataEl)
WriteVariable
retRteWrite = RTE_E_OK;
retE2EProtect = E2E_E_OK;
plausibilityChecks = E2E_E_OK;
AppDataEl == NULL
[TRUE]
[FALSE]
WriteVariable
plausibilityChecks =
E2E_E_INPUTERR_NULL
SerializeAppDataEl to IPDU representation
Data if AppDataEl is complex data element;
else it is a byte array, copy it to Data.
retE2EProtect = E2E_PXXProtect
(Config, State, Data)
(::)
retE2EProtect == E2E_E_OK
[FALSE]
[TRUE]
Write back CRC and
Counter from Data to
AppDataEl
retRteWrite = Rte_Write_<p>_<o>
(AppDataEl)
(::)
WriteVariable
ret = retRteWrite |
(retE2EProtect<<8) |
(plausibilityChecks<<16)
return ret

Figure 12-18: E2EPW_Write activity diagram

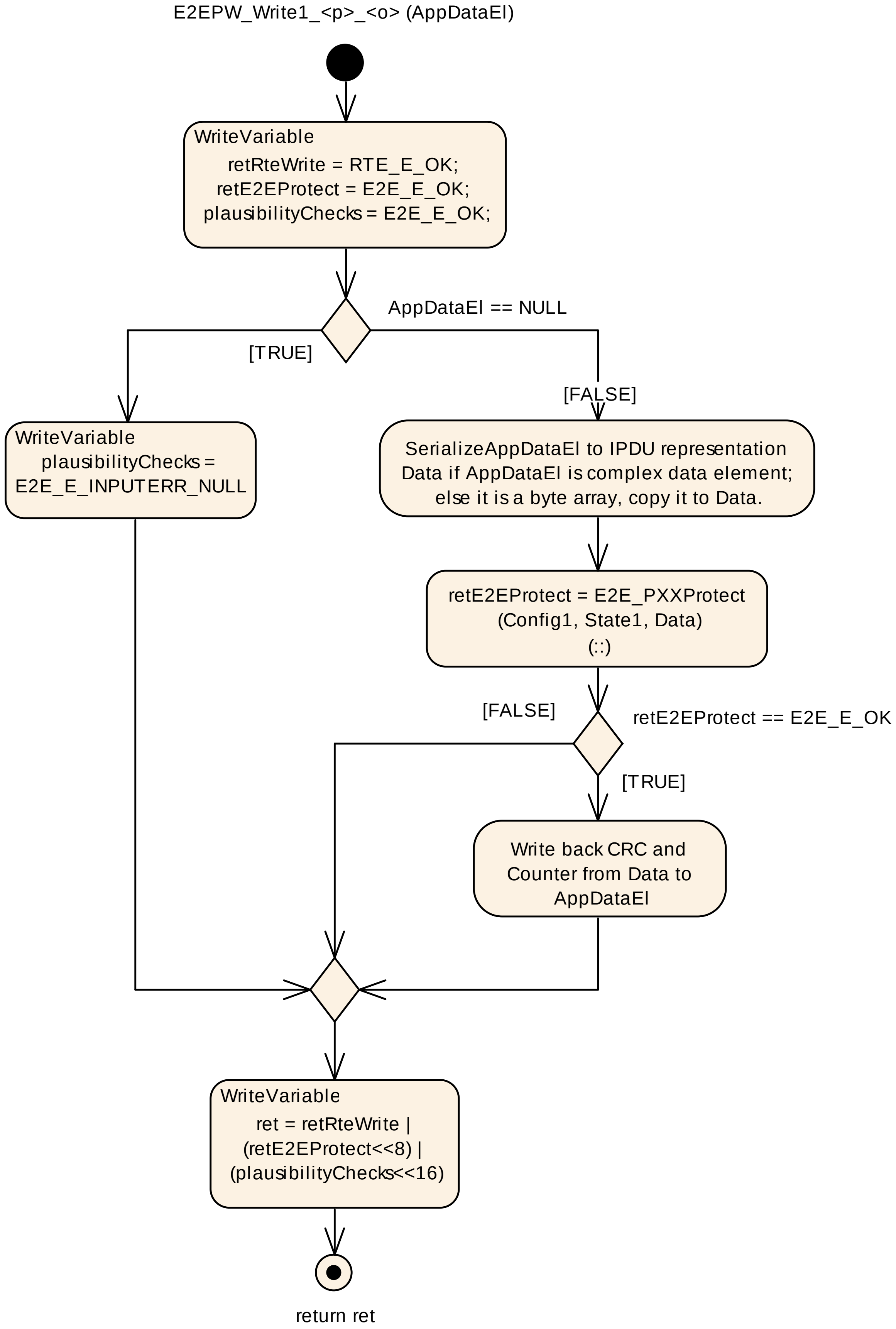


Figure 12-19: E2EPW_Write1 activity diagram

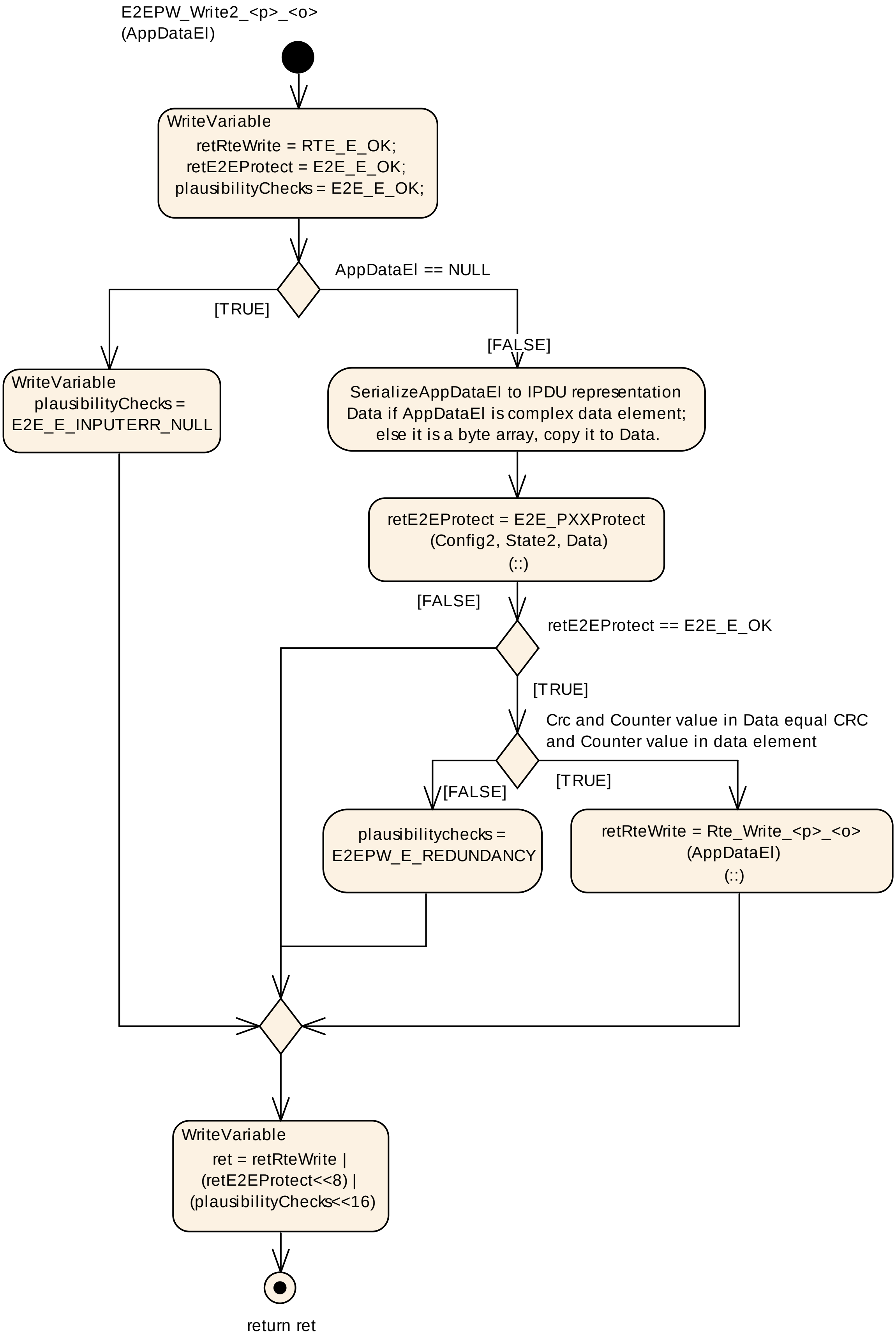


Figure 12-20: E2EPW_Write2 activity diagram

12.1.9 Code Example

Important:

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

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

In the code example, the suffix is formatted like this: ***_<p>_<o>***
This is to emphasize that <p> and <o> are placeholders.

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

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

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

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

1. Code redundancy:
   a. For each Rte_Write* function, there are corresponding E2EPW_Write1* and E2EPW_Write2* functions
   b. For each Rte_Read* function, there are corresponding E2EPW_Read1* and E2EPW_Read2* functions
2. Time diversity:
   a. The functions E2EPW_Write1* and E2EPW_Write2* on the sender side and E2EPW_Read1* and E2EPW_Write2* 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).

12.1.9.1 Code Example – Sender SW-C

12.1.9.1.1 Sender – E2EPW_WriteInit, E2EPW_WriteInit1 and E2EPW_WriteInit2

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

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

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

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

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

For redundant wrapper:

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


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


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


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


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

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

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

```
}
```

12.1.9.1.2 Sender – E2EPW_Write and E2EPW_Write1

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

12.1.9.1.2.1 Generation / Initialization

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

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

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

Case B (array):
The RTE Generator generates an opaque uint8 array.

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

12.1.9.1.2.2 Step S0

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

Case A (complex data type)

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

Case B (array):

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

```
AppDataEl [4] = 0xFF;
```

12.1.9.1.2.3 Step S1

Step S1: Application calls E2E Protection Wrapper.

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

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

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

12.1.9.1.2.4 Step S2

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

Case A (complex data type):
The E2E Protection Wrapper (**E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()**) serializes the data to the layout identical with the layout of the corresponding signal group in the I-PDU. It fills in unused bits with a predefined pattern, e.g. ‘1’-s (as defined in unusedBitPattern of 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_*<p>*_*<o>*** array contains the representation of only one signal group mapped to the I-PDU.

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


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

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

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

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

/* low nibble of high byte goes to Data[3] */
```

```
Data_<p>_<o>[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_<p>_<o>[3] |= 0xF0;

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

The above example is illustrated by the figure below:

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

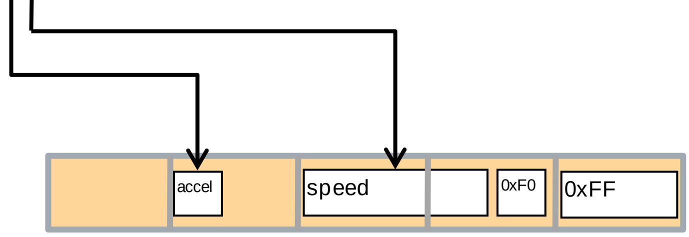


Figure 12-21: Mapping of Data elements into I-PDU

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

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

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

12.1.9.1.2.5 Step S3

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

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

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

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

12.1.9.1.2.6 Step S4

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

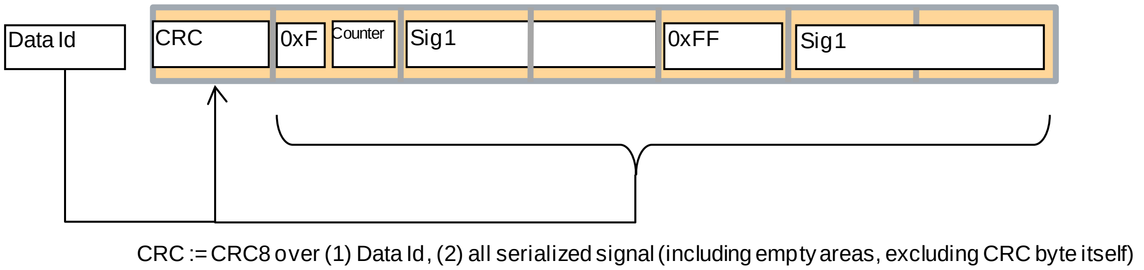


Figure 12-22: Step 4

12.1.9.1.2.7 Step S5

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

<u>Case A (complex data type):</u>

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

This is illustrated by the Figure 12-23:

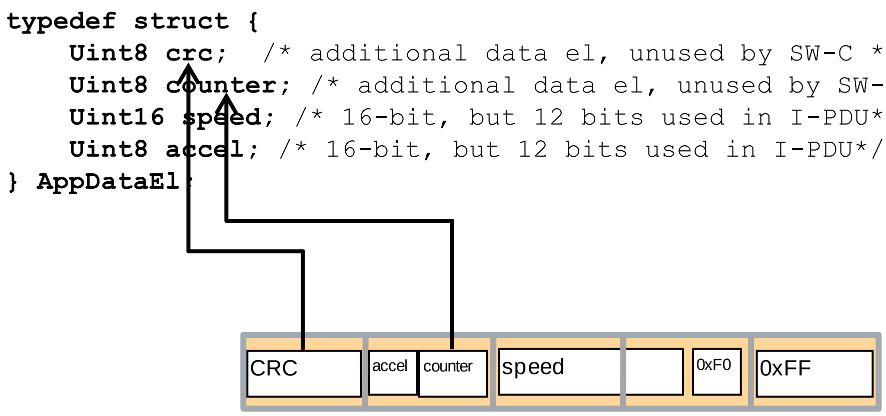


Figure 12-23: Copy back of CRC and alive from I-PDU copy to data element

<u>Case B (array):</u>

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

12.1.9.1.2.8 Step S6

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

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

Redundant wrapper (**E2EPW_Write_<p>_<o>**) in step S7 does *not* call **Rte_Write_<p>_<o>()** function.

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

12.1.9.1.2.9 Step S7

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

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

12.1.9.1.2.10 Step S8

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

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

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

12.1.9.1.3 Sender - E2EPW_Write2

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

12.1.9.1.3.1 Step S10

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

Step S10-S19 are only for the redundant scenario. The step S10 is just the repetition of S0 on the same values. The application rewrites the data in AppDataEl. 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 **AppDataEl** (e.g. corruption from outside, random memory fault on that area)

12.1.9.1.3.2 Step S11

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

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

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

12.1.9.1.3.3 Step S12

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

12.1.9.1.3.4 Step S13

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

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

12.1.9.1.3.5 Step S14

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

12.1.9.1.3.6 Step S15 – skipped

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

12.1.9.1.3.7 Steps S16

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

1. **AppDataEl** 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 **AppDataEl** filled in in step S0.
2. **Data** containing:
    a. **crc** and **counter** filled in by **E2EPW_Write2_<p>_<o>()**, based on **AppDataEl** from Step S10.

There are two safety mechanisms provided:

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

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

Case A (structure):

```
if(   (AppDataEl->counter != (Data_port1_de1[1] & 0x0F)) ||
          (AppDataEl->crc != (Data_port1_de1[0]        ))    )
      plausibilityChecks = E2EPW_E_REDUNDANCY; /* 0x05 */


Std_ReturnType retRteWrite = E2E_E_OK;

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

Case B (array):

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

Std_ReturnType retRteWrite = E2E_E_OK;

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

12.1.9.1.3.8 Step S17

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

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

12.1.9.1.3.9 Step S18

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

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

12.1.9.2 Code Example – Receiver SW-C

12.1.9.2.1 Receiver - E2EPW_ReadInit, E2EPW_ReadInit1 and E2EPW_ReadInit2

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

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

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

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

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

For redundant wrapper:

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

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

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

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

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

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

Std_ReturnType E2EPW_ReadInit2_<p>_<o>(Rte_Instance Instance) {
  State2_<p>_<o>.LastValidCounter = 0;
  State2_<p>_<o>.MaxDeltaCounter = 0;
  State2_<p>_<o>.WaitForFirstData = TRUE;
  State2_<p>_<o>.NewDataAvailable = FALSE;
```

```
  State2_<p>_<o>.LostData = 0;
  State2_<p>_<o>.Status = E2E_P01STATUS_NONEWDATA;
  State2_<p>_<o>.SyncCounter = 0;
  State2_<p>_<o>.NoNewOrRepeatedDataCounter = 0;
  return E2E_E_OK;
}
```

12.1.9.2.2 Receiver - E2EPW_Read and E2EPW_Read1

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

12.1.9.2.2.1 Generation / Initialization

Generation/Initialization: RTE generates a complex data element (case A) or an opaque uit8 array (Case B).

<u>Case A (complex data type):</u>
The RTE Generator generates the complex data element for the receiver. The complex data element has additional two data elements crc and counter, which are unused by SW-C application part, but only by the E2E Protection Wrapper. The data element is the same on the sender and on the receiver SW-C.

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

<u>Case B (array):</u>
The RTE Generator generates an opaque uint8 array.

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

12.1.9.2.2.2 Step R1

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

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

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

12.1.9.2.2.3 Step R2

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

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

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

12.1.9.2.2.4 Step R3

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

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

Redundant wrapper:

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

12.1.9.2.2.5 Step R4

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

Case A (complex data type):

```
/* For storing the same layout as the one of I-PDU */

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

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

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

/* low nibble of high byte goes to Data[3] */
Data_<p>_<o>[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_<p>_<o>[3] |= 0xF0;

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

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

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

12.1.9.2.2.6 Step R5

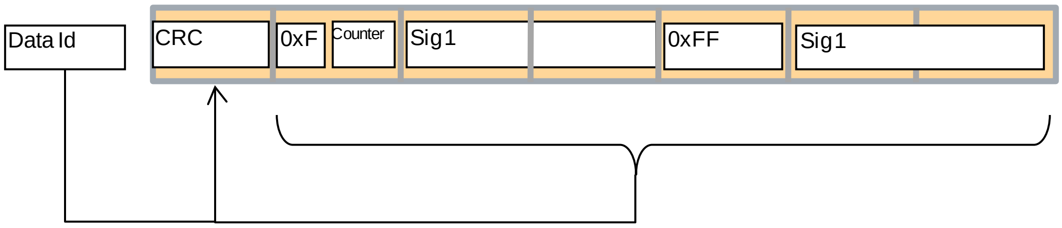
Step R5: E2E Protection Wrapper calls the E2E library to check the data element.

```
/* single channel - Read */
Std_ReturnType retE2ECheck = E2E_P01Check(&Config_<p>_<o>,
&State_<p>_<o>, Data_<p>_<o>);
```

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

```
/* redundant - Read1 */
Std_ReturnType retE2ECheck = E2E_P01Check(&Config1_<p>_<o>,
&State1_<p>_<o>, Data_<p>_<o>);
```

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



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

12.1.9.2.2.7 Step R7

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

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

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

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

/* in speed, only 8+4 bits are used.
   Topmost 4 bits shall be 0 */
if( (AppDataEl->accel & 0xF000) != 0)
      plausibilityChecks = E2EPW_E_DESERIALIZATION;
```

Case B (array):

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

```
plausibilityChecks = E2E_E_OK;
```

12.1.9.2.2.8 Step R8

Step R8: The E2E wrapper returns to the application.

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

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

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

12.1.9.2.2.9 Step R9

Step R9: Caller SW-C checks the return value and handles errors, if any. This behavior is specific to the application. Then it copies the data from **`AppDataEl`** to application buffer and consumes it.
Note that the caller may accept some errors on byte 3 (e.g. it may accept if byte 3 equals to E2E_PXXSTATUS_OKSOMELOST).

Case A (complex data type):

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

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

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

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

Case B (array):

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


uint16 targetSpeed = (AppDataEl[2]) | (AppDataEl[3]<<8 & 0x0F);
uint8 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(wrapperRet1);
}

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

12.1.9.2.3 Receiver - E2EPW_Read2

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

12.1.9.2.3.1 Step R10 – skipped

Value unused to numbering consistency.

12.1.9.2.3.2 Step R11

Step R11: Application calls the wrapper again.

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

12.1.9.2.3.3 Step R12 – partially skipped

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

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

12.1.9.2.3.4 Steps R13

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

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

12.1.9.2.3.5 Steps R14

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

12.1.9.2.3.6 Step R15

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

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

12.1.9.2.3.7 Step R16

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

12.1.9.2.3.8 Step R17

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

12.1.9.2.3.9 Step R18

Step R8: The E2E wrapper returns to the application.

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

12.1.9.2.3.10 Step R19

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

Case A (complex data type):

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


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

/* Check if both Read1 and Read2 report the same status.
   In particular, byte2 of ret1 and ret2 shall be identical. If not,
   then it means that there is a disagreement on evaluation
   of data between Read1 and Read2 */
if(wrapperRet2   !=   wrapperRet1)   swc_error_handlerR(wrapperRet1,
wrapperRet2);


/* check for corruption of AppDataEl after CRC has been checked */
if(targetSpeed2  !=  targetSpeed1)  swc_error_handlerR(wrapperRet1,
wrapperRet2);
if(targetAccel2  !=  targetAccel1)  swc_error_handlerR(wrapperRet1,
wrapperRet2);

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

Case B (array):

```
/* copy values from data element */
uint16 targetSpeed2 = (AppDataEl[2]) | (AppDataEl[3]<<8 & 0x0F);
uint8 targetAccel2 = AppDataEl[1] >> 4;


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

/* Check if both Read1 and Read2 report the same status.
   In particular, byte2 of ret1 and ret2 shall be identical. If not,
   then it means that there is a disagreement on evaluation
   of data between Read1 and Read2 */
if(wrapperRet2   !=   wrapperRet1)   swc_error_handlerR(wrapperRet1,
wrapperRet2);
```

```
/* check for corruption of AppDataEl after CRC has been checked */
if(targetSpeed2  !=  targetSpeed1)  swc_error_handlerR(wrapperRet1,
wrapperRet2);
if(targetAccel2  !=  targetAccel1)  swc_error_handlerR(wrapperRet1,
wrapperRet2);

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

12.2 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.

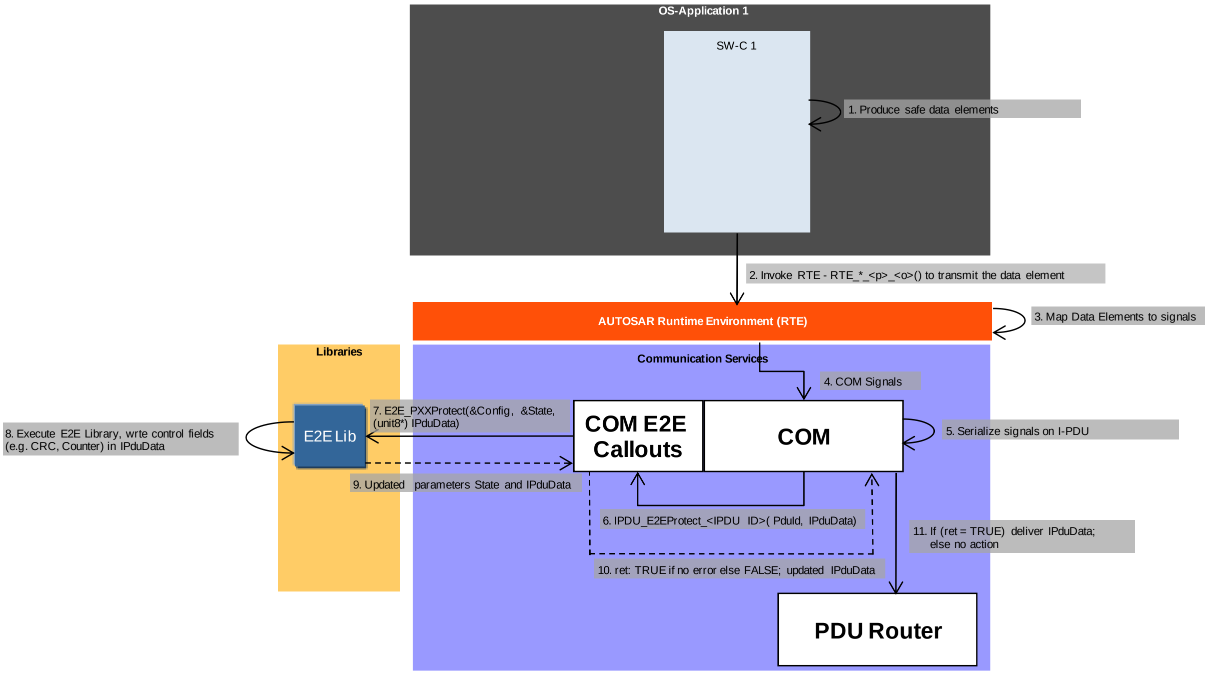


Figure 12-24: Callout – overall flow – P-port

The diagram below summarizes the COM E2E Callout solution. The very important step is that the E2E Library overwrites CRC byte in the signal group by the check status bits (E2E_PXXCheckStateType). Then, this overwritten CRC byte is converted by COM to signals and then by RTE to data elements. As a result, the SW-C receives in the CRC data element the E2E check bits, and not the CRC value.

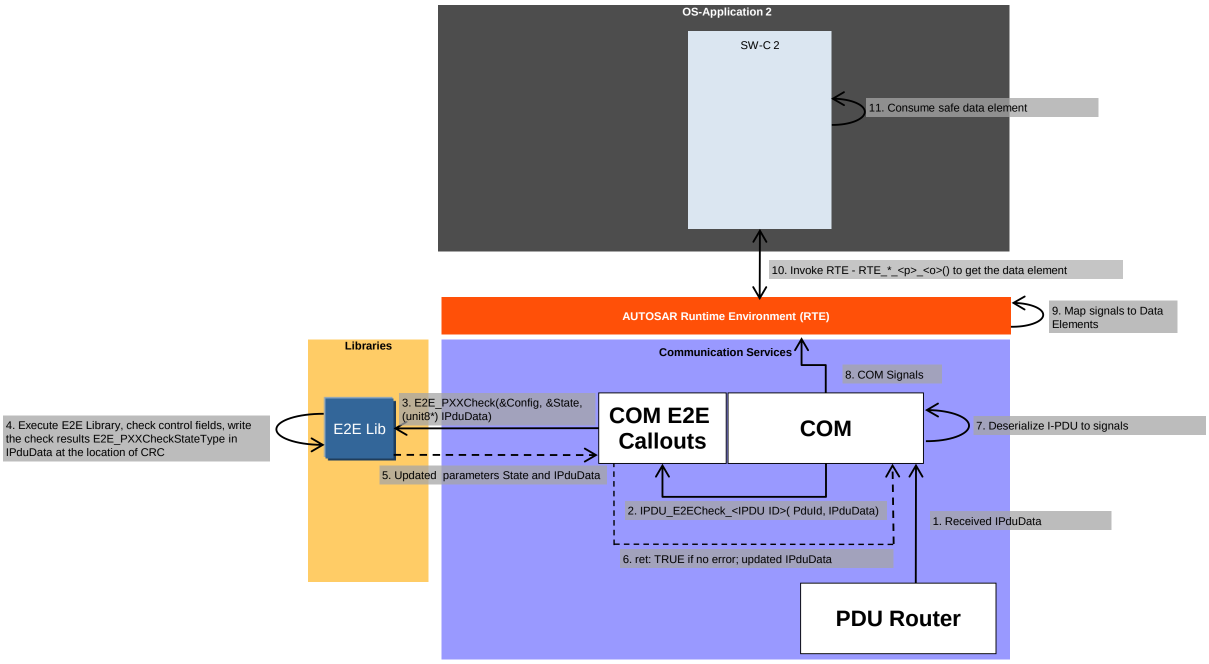


Figure 12-25: Callout – overall flow – R-port

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 (fuction 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.

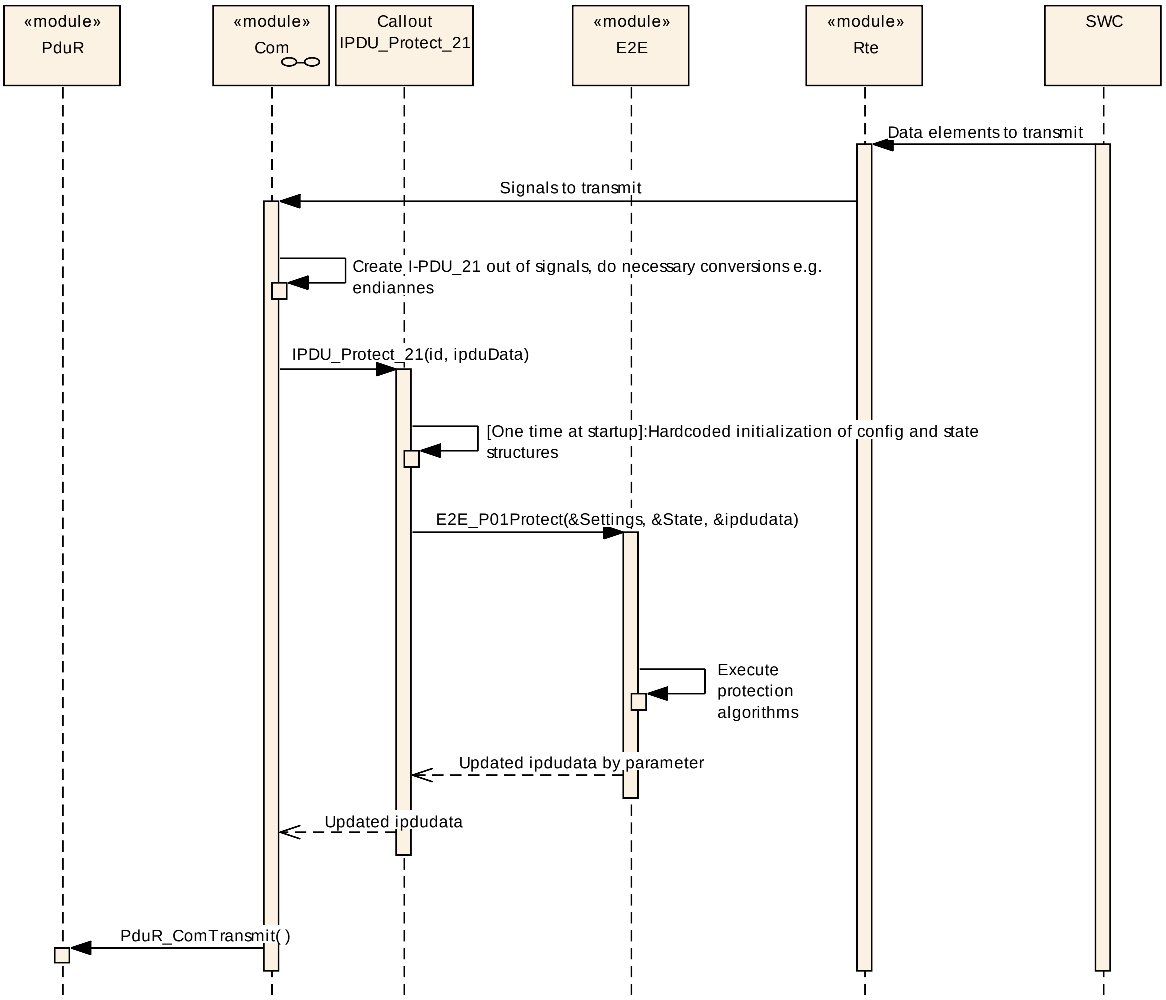


Figure 12-26: Callout – sequence – sending

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().`⌋ (SRS_E2E_08528)

Reception

On the receiver COM side, when the I-PDU is available at PDU Router, PDU Router invokes COM's function COM_RxIndication(). COM then calls the generated I-PDU callout (if configured for the given I-PDU). The callout, generated specifically for that I-PDU, calls the E2E Library with specific parameters (once for each E2E-protected signal group). The E2E Library executes the checks and stores the check results in the status.
Once E2E Library check function returns, the callout copies the status into the CRC byte, so that it can be analyzed, if needed, by receiver SW-C.

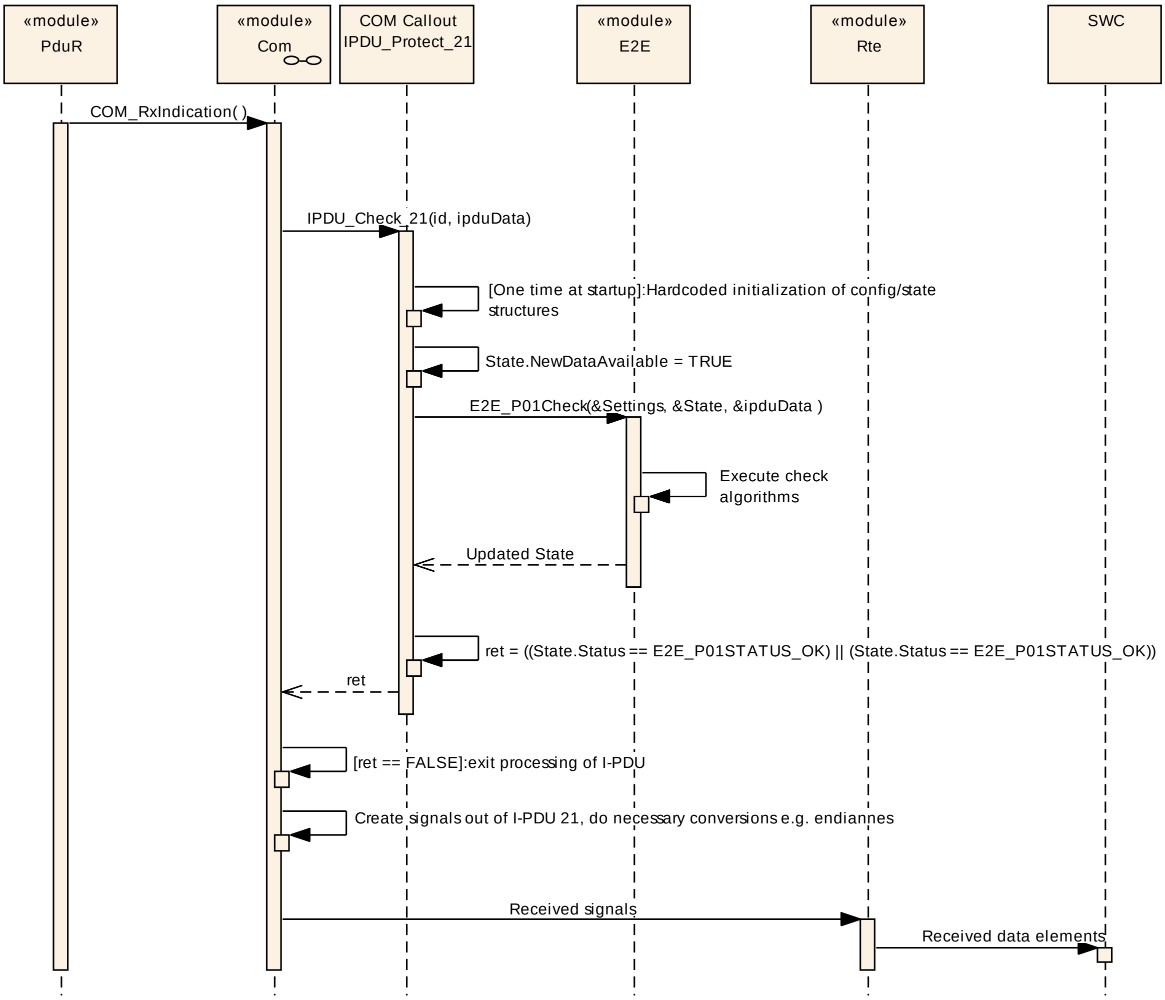


Figure 12-27: Callout - sequence - reception

[UC_E2E_00251]⌈ The reception callout for usage with E2E shall be the following: `IPDU_E2Echeck_<IPDU ID>(SRS_E2E_08528)`.

For example, the callout to protect the signal groups in an I-PDU with handle 21 shall have the name `IPDU_E2ECheck_21()`.⌋ (SRS_E2E_08528)

12.2.2 Methodology

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

The information how each signal group needs to be protected (e.g. which E2E Profile, which offset) is defined in System Template [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 identifers) 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 genrated, 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.⌋ (SRS_E2E_08528)

[UC_E2E_00290]⌈ If the E2EProtection is done via COM Callouts then the EndToEndProtectionISignalIPdu shall be defined.⌋ (SRS_E2E_08528)

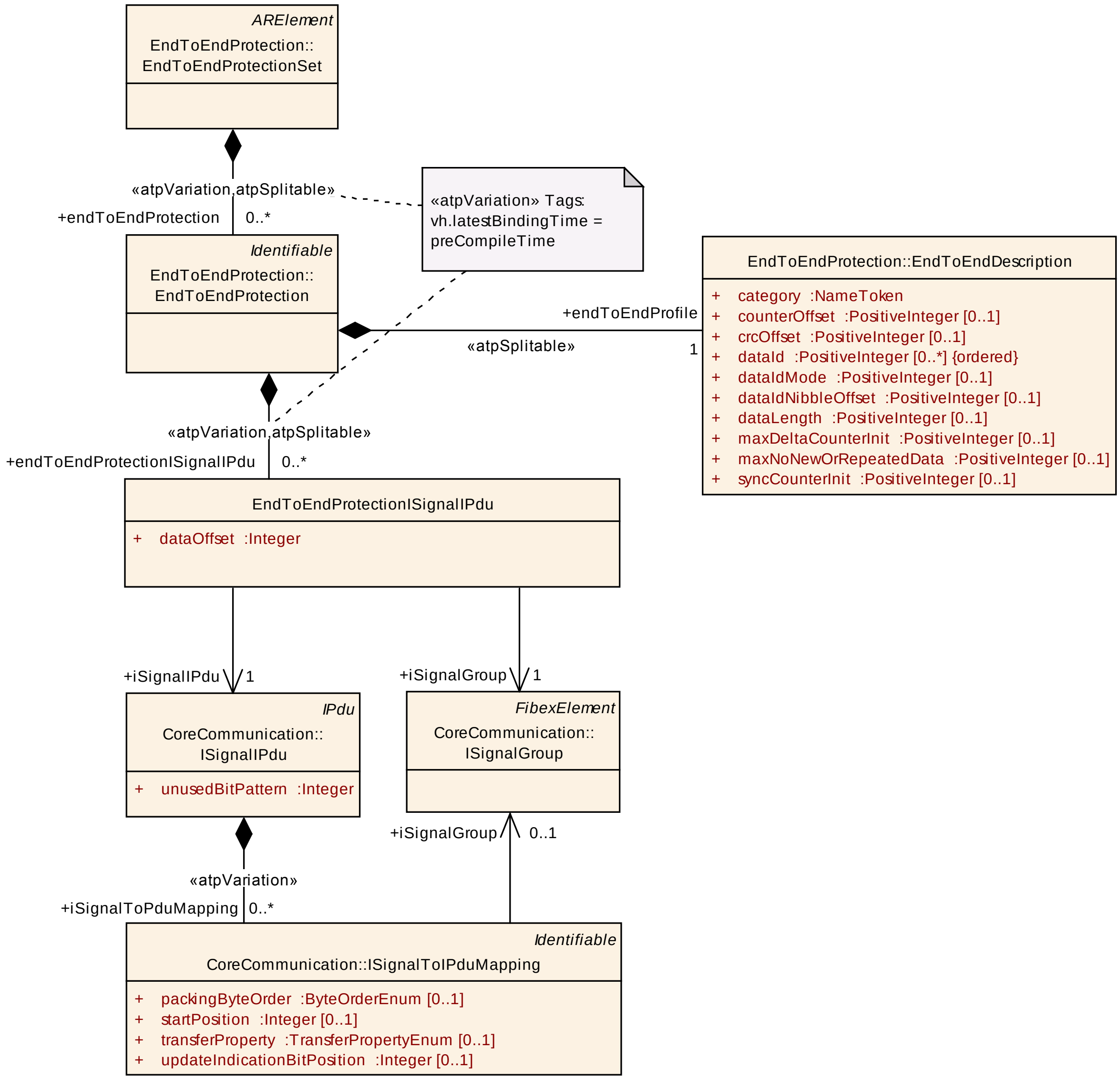


Figure 12-28: Release R4.0.1 and newer: COM Callouts Configuration (hardcopy from DOC_PduEndToEndProtection)

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. 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.

ISignalIPdu.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 <= M <= 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_P01ProtectStateType 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_P01CheckStateType Sta_Read_21 =
                         {0, 0, TRUE, FALSE, E2E_P01STATUS_NONEWDATA };

     /* If callout is invoked, this means that new data is available
```

```
      At COM */
      Sta_Read_21.NewDataAvailable = TRUE;

      Std_ReturnType ret = E2E_P01Check(Cfg_Read_21, Sta_Read_21, 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 Protection at RTE level through E2E Transformer

In this scenario, the RTE is considered safety-related. COM is QM. The RTE does the serialization of data elements into one dynamic-size signal, then RTE calls E2E to protect it. Then, RTE provides this E2E-protected dynamic-size signal to COM.

This solution is out-of-box, which means that AUTOSAR needs to be configured, but there is no need of integrator code for the E2E invocation.

This scenario is specified in details in SWS E2E Transformer.

principles are provided in this chapter.independently.

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 1st →	6	18	**6**	**18**	**6**	**18**	=	18	erroneously undetected! (valid)
routing error	7	164	**7**	**200**	**7**	**200**	≠	164	error detected (not valid)
	8	59	**8**	**131**	**8**	**131**	≠	59	error detected (not valid)
	9	205	**9**	**62**	**9**	**62**	≠	205	error detected (not valid)
	10	100	**10**	**244**	**10**	**244**	≠	100	error detected (not valid)
	11	246	**11**	**175**	**11**	**175**	≠	246	error detected (not valid)
	12	141	**12**	**106**	**12**	**106**	≠	141	error detected (not valid)
	13	36	**13**	**37**	**13**	**37**	≠	36	error detected (not valid)
	14	182	**14**	**219**	**14**	**219**	≠	182	error detected (not valid)
	15	77	**15**	**150**	**15**	**150**	≠	77	error detected (not valid)
	…	…	…	…	…	…	…	…	…
	5	123	**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.

	For each value of counter: **DataID value to be used**																For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**																																								
#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
1	28	8	73	140	234	86	71	199	14	46	36	103	158	122	129	242	X		X	X	X	X	X		X	X		X	X		X		X	X			X				X	X			X	X		X	X	X			X	X	X	X	X
2	180	134	53	233	243	121	67	176	9	94	168	132	198	142	201	138	X				X	X	X		X		X	X			X	X	X	X		X	X		X	X	X	X	X		X			X			X	X	X			X	X
3	78	206	124	3	132	58	100	23	154	160	136	220	127	93	37	11		X			X	X			X	X		X		X	X		X	X			X	X		X	X			X			X		X	X	X	X		X	X		X
4	158	161	150	33	238	151	203	19	139	104	55	2	184	16	105	67	X			X	X	X	X	X				X	X	X			X		X	X		X	X	X			X		X	X		X			X		X	X			X
5	137	17	243	195	157	94	103	59	201	1	187	183	11	159	143	146	X	X	X		X			X	X			X	X	X	X	X	X	X		X		X		X			X	X	X	X			X	X		X	X			X	
6	199	242	159	165	168	194	21	90	20	167	109	136	19	69	217	119		X		X			X		X			X	X	X	X	X					X	X	X	X	X		X		X	X	X		X		X	X			X		X
7	12	243	207	94	6	201	30	183	35	143	78	190	188	108	16	38	X	X		X				X	X		X	X	X	X		X		X	X			X	X	X	X				X	X	X	X				X	X		X		X
8	15	209	107	99	158	116	34	148	215	161	231	111	156	207	150	221		X	X	X		X	X			X		X			X	X	X		X		X					X	X	X	X	X	X			X	X		X	X	X		
9	245	53	92	121	207	9	230	132	113	201	5	47	4	179	35	160	X	X	X		X	X	X	X	X	X	X	X	X	X		X		X		X	X	X	X				X		X	X	X	X	X		X	X	X	X	X	X	
10	4	65	1	42	110	208	168	45	211	25	191	226	12	167	122	106	X	X		X	X					X			X		X	X				X	X	X	X	X	X	X	X			X		X	X	X	X		X	X		X	X
11	125	130	213	139	180	235	221	187	162	134	199	78	95	229	53	123		X	X	X	X	X		X	X	X	X			X	X	X	X		X		X	X		X	X	X				X			X	X	X	X	X	X	X		
12	159	194	20	136	217	173	214	95	89	144	158	77	230	186	51	205	X	X	X			X	X	X	X	X	X	X					X	X		X						X	X	X			X			X	X	X		X	X		X
13	246	35	39	72	84	175	138	251	13	135	106	15	71	120	49	79		X		X	X	X		X		X			X	X	X				X	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X		X		X	X
14	186	170	140	32	215	199	25	141	103	102	16	242	163	113	91	114			X		X	X	X	X	X		X		X			X	X			X	X		X	X	X	X	X	X	X	X			X		X		X	X	X		
15	29	68	44	31	8	152	1	56	19	140	246	219	209	208	86	247	X	X	X	X		X	X	X	X	X		X			X	X		X	X	X	X		X	X	X	X	X		X	X	X	X	X		X	X		X	X	X	X
16	149	178	48	27	29	185	17	203	148	68	204	36	101	195	44	16	X		X		X							X			X	X		X	X	X			X		X		X	X		X		X	X		X	X	X	X	X	X	X
17	250	60	120	89	66	216	164	137	202	112	125	12	206	73	74	17			X			X				X	X		X	X	X	X			X	X	X	X	X	X	X	X			X		X	X			X		X	X		X	
18	160	145	248	202	149	236	182	34	83	178	198	196	106	105	48	207		X	X			X					X	X	X	X		X	X		X	X		X			X		X		X			X	X	X	X	X	X	X	X	X	
19	42	45	226	106	113	62	80	154	133	115	60	98	94	63	21	212	X	X	X	X	X	X		X	X	X	X	X	X	X			X	X	X	X	X			X	X		X		X	X	X	X	X		X		X	X	X		
20	73	86	14	103	129	30	118	159	191	193	229	108	150	131	127	194	X	X			X	X	X		X	X	X			X	X	X			X	X		X		X	X		X				X		X	X		X	X	X	X	X	
21	203	16	84	189	56	135	146	117	49	247	47	109	88	66	232	85		X	X	X	X			X	X	X	X	X		X	X	X	X			X	X	X	X			X			X		X	X		X		X	X			X	
22	23	11	188	245	202	204	151	53	110	34	129	92	51	2	196	121		X	X	X		X	X	X	X			X	X	X		X	X	X	X	X			X		X	X	X	X	X	X				X		X		X			X
23	136	95	77	205	225	52	170	129	17	198	33	10	128	32	4	193			X		X	X	X		X		X		X				X		X	X		X		X		X			X	X	X		X	X			X	X	X		
24	45	106	62	154	115	98	63	212	227	101	89	24	183	241	69	171	X	X	X	X			X	X	X		X				X	X			X				X	X	X			X	X		X		X	X	X	X	X	X		X	X
25	126	6	203	143	54	16	127	211	84	240	121	189	236	210	56	224		X		X	X		X	X	X	X	X	X		X		X	X	X	X	X		X	X	X	X		X	X	X		X	X		X	X		X	X	X		X
26	39	175	13	15	49	40	145	209	184	197	98	107	118	202	100	99		X	X			X				X	X			X	X	X	X	X		X			X	X	X		X		X	X	X		X		X	X			X		X
27	114	177	90	184	43	119	50	223	95	237	29	234	232	83	63	215	X	X		X		X	X	X		X		X	X			X		X		X				X	X		X		X	X	X		X	X	X	X		X		X	
28	148	221	7	229	19	81	190	49	200	67	41	82	170	169	230	197	X	X	X	X	X	X		X		X	X		X	X	X	X	X	X	X	X		X		X	X		X	X	X	X	X			X	X						X
29	1	208	211	226	122	224	217	62	117	41	138	133	207	144	131	98	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X	X			X	X			X	X		X	X	X	X		X	X				X	X
30	88	218	87	39	221	57	132	175	114	229	226	13	54	160	81	15		X			X		X	X	X		X	X	X	X			X	X	X	X		X		X	X	X				X	X	X	X			X	X	X		X	X
31	107	116	215	111	150	102	246	7	91	151	240	244	28	35	139	81		X	X	X		X	X				X	X	X		X	X	X	X	X				X	X	X			X	X	X		X	X	X			X	X	X	X	
32	242	165	194	90	167	136	69	119	173	80	76	95	67	239	144	234	X		X	X				X	X	X		X	X			X		X				X		X		X		X		X	X	X					X	X	X		
33	37	248	50	236	79	83	180	196	249	48	4	88	62	134	99	246	X	X	X	X	X	X	X	X		X			X	X					X	X	X			X	X	X		X	X	X	X	X	X		X		X	X		X	
34	219	232	142	64	159	96	133	222	128	194	15	172	7	227	20	164		X		X		X	X	X	X			X	X	X		X	X	X			X		X	X	X	X		X	X	X	X		X		X	X			X		X
35	33	19	2	67	14	230	189	142	123	30	57	179	215	117	191	96	X		X	X	X			X	X		X			X		X	X				X	X	X	X	X	X	X		X	X	X		X		X	X	X	X		X	X
36	132	160	37	145	169	248	61	202	50	149	225	236	45	182	79	34		X			X	X			X		X				X	X	X	X	X				X		X	X	X	X	X	X	X	X	X			X		X	X		X
37	104	14	70	30	10	191	247	108	176	127	81	250	130	192	59	20	X	X	X	X	X		X	X		X		X	X	X			X	X	X			X	X	X	X	X		X	X					X	X			X	X		X
38	175	15	40	209	197	107	202	99	223	158	24	116	131	34	93	148	X	X	X		X		X	X	X	X	X	X	X	X	X	X			X		X	X	X	X	X	X		X		X		X	X		X		X	X	X	X	X
39	35	72	175	251	135	15	120	79	40	109	154	209	122	216	197	48		X	X	X	X	X		X	X	X	X	X	X	X			X	X	X	X		X	X	X	X		X			X	X	X		X	X	X		X	X		X
40	80	63	225	241	23	198	116	213	4	11	73	153	20	111	188	235	X	X	X			X		X	X		X	X		X		X		X	X			X	X	X			X	X	X			X		X	X	X	X	X	X		X
41	182	105	134	70	17	233	219	191	121	195	244	176	241	232	94	250		X		X	X	X		X		X			X	X	X	X		X	X				X					X			X	X		X	X						X
42	106	154	98	212	101	24	241	171	22	50	137	231	38	213	239	110		X		X	X	X	X		X	X	X	X			X		X	X		X	X		X	X		X	X			X	X	X	X	X	X	X		X	X		X
43	202	34	196	207	27	246	70	201	218	203	162	35	212	191	36	190	X	X	X		X					X	X	X	X	X	X	X	X	X	X	X	X			X			X	X	X	X	X		X	X		X	X		X		
44	163	44	33	152	73	19	143	219	2	86	218	67	107	211	14	232	X	X			X	X	X	X	X		X	X			X	X	X	X		X		X	X	X		X	X	X	X	X	X	X	X		X	X	X				
45	19	67	230	142	30	179	117	96	97	108	13	128	102	85	250	172	X	X	X	X	X	X	X		X	X					X	X					X		X	X		X	X	X	X		X	X		X		X		X			X
46	112	181	126	157	24	6	129	1	203	231	134	143	145	193	54	208	X		X	X	X		X			X	X	X		X	X	X				X		X					X	X	X	X				X	X		X	X	X	X	
47	118	131	192	156	114	166	154	214	222	177	149	28	84	212	90	186		X		X		X		X		X			X	X		X			X		X			X	X	X	X					X	X	X	X	X		X	X	X	X
48	247	192	64	166	165	222	98	28	172	90	79	164	229	24	136	8		X	X	X	X		X	X	X	X	X			X				X		X	X	X		X		X		X	X	X	X	X		X	X	X		X	X	X	
49	50	83	249	88	99	55	53	218	32	148	1	87	227	121	111	39	X			X	X	X				X	X		X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X		X	X	X	X	X				X	X	X
50	70	191	176	250	59	138	64	60	160	146	43	120	235	222	38	89		X	X	X	X	X	X				X		X		X	X		X		X	X			X	X	X		X	X		X	X		X			X	X	X	X	

For each value of counter: **DataID value to be used**

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
51	251	79	209	48	76	99	12	185	116	163	171	148	21	243	161	36
52	200	43	80	237	97	63	209	125	225	75	28	241	159	99	23	130
53	188	204	110	92	196	25	187	9	122	246	127	113	181	123	218	47
54	87	57	114	13	81	177	37	40	90	82	133	184	56	248	43	107
55	122	41	131	21	57	156	58	69	166	13	18	214	16	220	177	239
56	53	121	9	132	201	47	179	160	115	190	217	37	65	128	72	145
57	121	132	47	160	190	37	128	145	101	169	144	248	42	61	251	202
58	204	92	25	9	246	113	123	47	41	35	210	115	157	97	39	37
59	89	137	12	17	181	243	86	195	207	157	139	94	23	103	6	59
60	174	124	106	58	47	154	237	220	98	37	173	212	59	125	101	188
61	157	1	143	208	71	211	5	226	189	122	176	224	34	217	118	62
62	144	225	26	198	220	4	215	153	157	188	104	65	60	102	171	162
63	117	85	76	29	166	163	181	68	161	28	236	44	175	157	164	31
64	146	66	18	112	62	74	95	181	178	98	228	126	132	205	227	157
65	115	101	69	50	15	239	11	83	186	209	181	249	224	245	107	88
66	3	23	220	11	145	188	150	245	171	202	205	204	147	151	236	53
67	24	231	54	71	249	240	162	122	56	55	94	118	178	174	32	41
68	172	238	182	104	77	105	31	14	134	52	111	70	237	56	17	30
69	11	245	204	53	34	92	2	121	25	207	193	9	26	230	246	132
70	197	158	93	161	172	150	27	33	130	238	249	151	177	203	182	19
71	213	235	162	78	53	174	81	206	45	121	159	124	52	82	9	3
72	131	156	166	214	177	28	212	186	164	184	178	8	135	171	119	170
73	212	171	231	110	83	71	235	25	240	88	195	122	74	78	55	113
74	9	47	115	37	72	101	75	248	69	251	51	50	208	228	15	236
75	173	77	137	52	26	17	140	10	243	4	151	195	75	199	157	127
76	192	166	222	28	90	164	24	8	238	119	48	73	49	231	95	140
77	116	111	102	7	151	244	35	81	168	2	118	200	8	72	187	82
78	123	97	3	75	138	23	158	228	220	120	119	11	5	161	145	180
79	46	91	193	168	235	5	57	167	210	78	232	217	104	13	174	80
80	151	2	187	230	70	123	135	179	206	191	114	97	46	109	176	128
81	229	49	82	197	142	100	149	158	237	96	69	93	141	178	128	161
82	21	69	214	239	40	186	188	249	8	107	126	170	117	204	223	55
83	47	37	101	248	251	50	228	236	239	79	26	83	226	180	209	196
84	128	61	228	182	89	180	44	105	245	137	215	134	80	152	12	70
85	76	163	161	44	164	33	6	152	151	73	88	19	40	143	104	219
86	240	118	247	131	141	192	106	156	64	114	169	166	36	154	165	214
87	183	38	169	18	224	149	173	74	79	133	23	178	9	77	85	126
88	169	149	79	178	85	48	137	27	99	29	188	185	115	17	163	203
89	248	236	83	196	48	88	134	246	55	185	65	218	98	233	148	35
90	67	142	179	96	108	128	85	172	75	20	40	61	244	29	60	238
91	55	87	141	57	7	114	47	13	165	81	224	177	31	37	200	40
92	209	99	116	148	161	111	207	221	102	33	71	7	214	201	151	229
93	49	197	100	158	96	93	178	161	125	172	239	150	114	27	61	33
94	93	150	130	151	182	139	36	2	235	105	32	187	119	84	134	230
95	138	120	145	216	18	202	238	12	236	74	241	34	124	104	178	243
96	189	117	109	85	192	76	112	29	158	166	248	163	39	181	222	68
97	224	133	85	227	156	29	26	22	163	214	202	68	72	4	28	54
98	22	54	31	240	170	56	42	118	219	32	201	247	48	45	199	131
99	6	143	16	211	240	189	210	224	135	118	132	117	196	147	247	133
100	208	226	224	62	41	133	144	98	85	21	120	227	201	225	156	24

For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**

#	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
51	X	X		X	X			X	X	X			X	X	X	X				X	X	X			X	X	X		X	X	X			X		X	X			X	X
52	X	X	X	X		X	X	X	X	X							X						X	X			X					X	X	X		X			X	X	X
53	X	X	X	X			X	X	X	X	X		X		X	X	X				X	X	X		X	X	X	X	X		X		X	X			X			X	X
54	X		X			X			X	X		X	X	X		X	X	X	X		X	X	X	X	X	X	X	X			X	X	X	X	X			X	X	X	X
55	X					X		X	X		X	X	X	X	X		X	X			X	X	X		X		X		X	X	X	X		X	X	X	X	X		X	
56		X		X	X	X	X			X		X	X	X	X		X	X			X	X		X			X		X	X	X	X		X		X	X		X	X	
57	X				X	X	X			X	X	X	X	X	X			X		X		X	X	X			X		X	X	X	X	X	X	X	X	X	X	X	X	
58	X		X	X		X		X		X	X		X			X		X		X		X	X	X	X	X		X		X		X		X	X		X		X	X	X
59	X	X	X	X	X	X		X	X	X				X				X		X		X	X	X	X		X				X	X	X	X	X	X					X
60	X		X	X		X		X			X		X	X	X					X		X	X	X	X	X	X				X			X	X	X				X	X
61		X		X		X	X	X	X	X	X	X	X	X			X			X	X	X	X	X	X	X		X			X	X	X				X		X	X	X
62	X	X		X		X	X	X	X		X		X	X		X	X	X	X	X		X		X			X		X		X	X	X	X			X	X	X		
63		X		X		X	X	X		X	X	X	X		X		X	X		X	X		X	X		X	X	X	X	X	X	X		X		X	X	X			X
64		X	X		X			X		X			X	X	X	X	X	X	X			X				X	X	X	X			X		X	X	X	X		X	X	X
65	X	X	X	X		X	X			X	X	X	X			X	X	X	X	X	X	X	X	X		X	X	X	X	X	X		X		X	X		X	X		X
66	X	X	X	X		X	X	X	X		X		X	X		X		X	X			X		X	X	X	X	X	X		X	X	X			X	X	X	X		
67	X		X	X				X			X	X	X	X	X		X		X	X		X	X	X	X	X	X	X	X		X	X	X					X	X		X
68		X	X	X			X	X	X	X	X	X	X		X	X	X	X	X	X	X		X	X		X	X	X	X	X	X		X	X	X	X		X	X	X	X
69		X		X	X	X	X	X	X		X	X	X	X	X	X		X	X	X			X	X	X	X		X	X	X	X	X	X		X	X	X	X		X	X
70	X		X		X		X	X		X		X		X	X	X		X	X	X			X	X	X		X	X		X	X		X	X	X		X	X			X
71	X		X			X			X	X	X			X		X			X	X	X		X			X	X	X	X				X		X	X		X	X	X	
72			X	X	X			X		X	X	X	X	X	X	X		X	X	X	X			X		X		X	X					X	X					X	
73	X	X	X	X		X		X	X		X								X	X		X	X	X	X	X	X	X	X	X	X	X	X				X	X			X
74	X		X			X		X	X	X	X	X	X		X				X	X	X			X		X	X		X		X	X	X	X	X			X	X	X	
75	X	X	X	X	X	X	X	X	X	X		X	X				X	X	X		X	X				X					X	X		X	X		X			X	X
76		X		X		X	X	X		X	X						X	X	X	X	X			X		X	X	X	X	X		X	X	X	X	X	X	X	X	X	
77	X	X		X	X		X	X		X	X		X	X		X	X	X	X	X		X	X		X		X	X	X		X	X	X		X	X		X		X	X
78		X		X		X		X		X		X	X	X	X	X		X	X		X	X	X		X	X				X	X	X		X		X	X	X			X
79	X		X			X	X			X	X	X	X			X		X		X	X			X		X	X		X	X				X				X	X		
80	X	X		X		X	X	X		X	X	X	X	X	X	X	X	X		X		X		X		X	X	X		X	X			X	X	X			X	X	X
81	X	X	X		X	X		X	X	X	X	X	X		X	X			X		X				X		X		X	X					X	X	X		X	X	X
82	X		X			X	X		X	X	X	X	X		X	X	X			X			X	X	X	X	X			X			X		X	X	X			X	X
83		X	X	X			X		X		X			X	X	X					X	X	X	X	X			X		X				X	X		X	X	X	X	X
84	X	X		X	X	X			X		X		X		X	X	X			X	X	X		X	X	X	X	X				X	X	X			X	X			
85	X		X	X				X	X	X	X		X	X		X	X		X	X								X	X		X	X	X		X	X	X		X	X	X
86		X							X	X	X		X	X		X	X	X	X	X		X	X			X	X	X		X		X	X	X	X	X	X	X	X	X	X
87	X			X	X		X	X	X	X	X	X			X		X	X		X	X	X		X	X	X	X			X	X	X			X	X					
88				X					X	X			X	X	X	X	X	X	X			X		X	X		X	X		X	X	X	X	X	X			X	X		
89		X			X			X	X			X	X	X		X		X	X	X	X	X	X		X	X	X	X	X	X			X	X	X	X		X	X	X	X
90		X	X	X	X	X		X				X		X	X			X				X	X		X	X	X	X		X	X	X	X		X	X	X	X	X		X
91	X	X		X	X		X				X	X				X		X			X	X	X	X		X		X		X	X	X	X	X	X	X				X	X
92				X	X	X	X	X	X	X	X	X			X	X	X	X	X	X	X		X		X	X		X					X		X						X
93							X	X			X	X	X	X	X	X			X	X		X	X	X	X				X		X		X		X			X			
94		X		X		X		X			X			X	X	X		X	X	X				X				X		X	X	X	X	X	X	X	X	X	X		X
95			X	X		X		X		X	X	X	X	X			X		X	X	X	X	X	X	X	X		X	X	X	X	X	X	X		X	X	X			X
96	X	X		X	X	X	X			X							X			X	X			X		X			X		X	X	X		X	X		X		X	
97		X		X	X	X	X		X	X	X		X				X				X	X						X	X		X	X	X			X	X		X		
98	X		X			X		X			X	X	X	X	X	X	X			X	X	X		X				X	X		X	X	X	X	X	X	X		X		
99							X			X	X	X	X	X			X	X		X	X	X	X	X		X		X	X	X	X	X		X	X			X	X	X	X
100		X	X	X	X	X	X	X			X	X	X	X		X		X	X	X	X		X	X	X	X			X		X		X	X				X	X	X	

For each value of counter: **DataID value to be used**

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
101	218	39	57	175	229	13	160	15	177	49	62	40	240	145	82	209
102	8	140	86	199	46	103	122	242	30	91	84	159	161	41	193	165
103	31	56	219	247	199	232	226	192	142	242	72	64	148	62	159	166
104	184	223	234	215	125	46	88	102	129	130	31	91	222	218	213	244
105	135	109	197	76	64	158	74	163	93	222	50	161	57	126	172	44
106	62	98	227	24	69	22	198	231	68	239	12	54	169	153	186	71
107	44	152	19	219	86	67	211	232	230	103	39	142	116	224	30	64
108	105	70	233	191	195	176	232	250	132	59	200	138	213	64	183	60
109	124	58	154	220	37	212	125	188	24	248	77	171	146	130	50	204
110	215	102	91	244	139	168	39	200	5	187	247	167	73	175	78	43
111	92	9	113	47	35	115	97	37	21	72	147	101	1	75	175	248
112	85	29	163	68	28	44	157	31	33	8	196	152	15	1	73	56
113	249	55	32	87	111	141	9	57	242	7	211	114	68	47	244	13
114	17	195	94	59	1	183	159	146	190	208	123	38	245	194	211	66
115	38	18	149	74	133	178	77	126	48	227	11	27	47	52	29	6
116	69	239	186	249	107	170	204	55	140	116	6	32	85	92	215	87
117	233	176	132	138	183	160	96	120	37	38	80	145	162	172	169	216
118	56	247	232	192	242	64	62	166	96	165	251	222	221	98	194	28
119	168	167	217	80	206	144	40	63	51	3	222	225	30	107	58	241
120	7	81	200	82	230	43	251	100	80	179	156	237	199	79	97	93
121	113	115	21	101	175	69	23	50	214	15	112	239	211	11	40	83
122	170	32	199	141	102	242	113	114	159	244	189	165	44	115	168	177
123	108	20	60	173	147	89	28	77	216	51	93	137	123	8	112	52
124	94	183	190	38	211	169	20	18	251	224	3	149	92	173	117	74
125	194	136	173	95	144	77	186	205	137	225	161	52	179	170	26	129
126	217	144	51	225	58	26	223	198	181	220	238	4	250	215	212	153
127	216	12	34	243	126	207	14	94	246	6	235	201	220	30	203	183
128	171	110	71	25	88	122	78	113	118	218	59	41	126	206	87	115
129	211	224	117	133	131	85	51	227	76	156	145	29	35	26	166	22
130	141	114	165	177	200	90	101	184	136	43	85	119	219	50	80	223
131	20	173	89	77	51	137	8	52	12	26	150	17	97	140	181	10
132	206	3	58	23	160	220	93	11	212	145	95	188	210	150	248	245
133	129	193	127	5	162	210	114	217	146	174	142	147	105	177	45	144
134	201	190	72	169	189	251	60	149	15	117	58	79	25	89	109	178
135	147	51	112	26	154	181	234	4	126	212	182	157	138	46	24	65
136	18	74	178	126	227	27	52	6	185	22	245	203	37	10	68	143
137	161	33	151	19	104	2	16	67	187	14	87	230	223	189	70	142
138	120	216	202	12	74	34	104	243	196	126	213	207	58	14	27	94
139	232	64	96	222	194	172	227	164	61	136	209	238	81	22	173	73
140	68	31	152	56	140	219	208	247	67	199	35	232	99	226	103	192
141	214	186	8	170	223	140	110	32	86	215	203	199	76	25	46	141
142	166	28	164	8	119	73	231	140	104	234	185	86	197	71	205	199
143	156	214	28	186	184	8	171	170	73	223	27	140	109	110	234	32
144	72	251	15	79	109	209	216	48	107	76	212	99	41	12	158	185
145	152	219	67	232	103	142	224	64	179	159	175	96	111	133	108	222
146	241	213	153	235	245	162	7	78	42	53	103	174	77	81	92	206
147	145	202	236	34	178	196	105	207	88	27	153	246	154	70	185	201
148	234	46	129	91	213	193	87	168	127	235	219	5	238	57	162	167
149	154	212	24	171	50	231	213	110	54	83	17	71	18	235	249	25
150	10	127	59	210	42	146	165	147	38	45	179	66	134	90	226	51

For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**

#	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
101	X	X	X	X	X	X		X	X				X	X	X	X		X			X	X	X		X	X										X		X		X	X
102	X		X		X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X				X	X	X	X	X	X	X	X	X	X	X			X			
103	X	X		X		X	X	X	X	X	X	X	X						X		X					X	X				X					X	X		X	X	X
104		X				X			X	X		X		X	X		X	X			X	X		X	X	X		X			X	X	X	X					X		
105	X	X	X			X	X		X	X	X		X				X		X			X	X	X	X		X	X	X							X	X		X	X	X
106		X		X	X	X				X	X	X	X	X	X		X	X	X			X		X	X						X		X	X	X	X		X	X	X	
107	X		X			X	X	X				X		X	X	X	X	X	X		X	X		X					X	X	X		X	X	X	X		X		X	X
108	X					X	X				X	X	X	X	X					X		X	X	X				X	X	X	X	X		X	X		X		X	X	
109		X	X	X	X	X	X	X						X		X		X	X			X	X	X	X	X	X	X	X	X	X	X	X		X				X		X
110	X	X	X					X	X		X	X	X		X				X		X		X		X	X	X	X	X		X	X		X	X	X	X	X	X	X	X
111	X		X	X	X	X	X	X							X	X	X	X		X	X		X		X	X	X	X	X	X	X	X	X	X			X				X
112		X	X		X			X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X				X		X	X	X			X	X	X	X	X		X
113	X		X	X	X	X		X	X	X		X	X			X		X	X				X	X	X		X	X	X	X	X	X	X		X		X	X	X	X	
114	X	X	X			X			X	X		X	X	X	X		X	X			X	X			X		X	X				X			X	X	X			X	
115	X		X	X		X		X	X				X		X	X			X		X						X	X		X	X		X			X		X			
116			X	X		X	X	X	X	X	X			X			X			X	X	X	X	X	X		X	X	X		X		X	X		X	X		X		X
117		X	X	X	X	X		X		X	X	X		X	X		X			X	X	X	X		X				X		X	X	X	X	X	X	X		X	X	X
118			X	X	X	X				X	X		X	X	X		X	X			X					X		X	X	X	X			X	X				X	X	
119		X		X	X		X	X	X	X	X	X		X		X	X	X		X	X	X		X	X					X	X	X	X			X	X	X	X	X	
120	X		X	X		X				X			X	X	X			X	X		X	X		X	X	X	X	X		X	X	X		X	X			X		X	X
121	X	X	X				X		X	X	X	X	X	X		X	X	X	X		X	X	X	X	X	X			X			X	X	X		X	X			X	X
122	X	X		X		X	X		X	X	X		X		X		X	X	X	X	X					X	X		X					X	X	X		X	X	X	X
123	X				X	X		X	X		X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X			X		X	X	X	X			X		X
124	X	X	X	X	X		X	X		X	X	X	X	X	X		X	X	X	X		X				X	X	X		X	X		X			X		X	X	X	X
125	X	X			X		X	X	X		X			X	X	X	X	X		X	X		X	X	X	X	X					X	X		X	X			X	X	X
126		X		X		X		X	X			X	X	X		X	X		X		X	X	X		X	X			X	X						X	X	X	X		X
127	X	X	X			X	X	X		X	X	X				X	X	X		X	X	X	X	X	X	X		X	X			X		X	X	X	X		X	X	X
128	X		X	X		X		X		X	X	X	X	X		X	X	X		X	X	X	X		X		X	X	X	X	X	X	X		X	X			X		X
129				X	X	X	X		X			X		X	X			X			X	X	X			X		X	X		X	X		X	X	X	X	X	X	X	X
130		X			X	X	X		X	X	X	X	X	X	X		X	X	X	X	X	X			X	X		X	X	X		X		X	X			X	X	X	X
131	X			X		X				X		X		X	X	X			X	X		X	X	X		X	X		X	X	X	X				X		X	X	X	
132		X	X	X		X			X	X	X	X	X					X	X	X			X	X	X		X	X		X			X	X	X			X			X
133	X		X			X			X	X	X	X	X				X	X	X	X	X	X	X			X	X	X	X	X		X			X		X	X	X	X	X
134	X			X		X	X	X		X				X			X	X	X		X	X		X	X		X			X	X		X	X	X		X	X		X	
135	X	X			X		X	X		X	X	X	X	X	X	X			X	X	X			X		X		X			X		X	X	X	X			X	X	X
136	X	X	X		X				X	X	X	X	X	X			X		X			X	X	X						X	X	X	X	X	X		X	X	X	X	X
137		X	X	X		X			X	X	X	X	X	X		X	X			X	X	X	X	X	X	X		X		X		X		X	X	X	X	X	X		
138	X		X		X				X		X	X	X	X		X	X	X	X	X		X	X					X	X	X		X	X				X		X	X	X
139		X		X		X		X			X	X	X	X			X	X		X	X	X	X	X		X			X	X		X			X	X	X		X	X	X
140	X	X	X	X	X	X	X		X		X	X		X	X	X	X		X			X	X	X		X	X	X		X				X	X			X	X	X	
141	X	X		X		X	X			X	X	X		X	X		X		X	X		X		X	X		X	X	X		X	X	X	X	X	X	X				X
142			X		X					X	X					X	X		X	X	X		X		X	X	X		X		X	X			X		X	X	X	X	X
143	X	X	X	X	X	X	X		X	X	X		X	X		X	X	X					X		X		X	X	X		X	X	X	X	X	X			X	X	
144	X	X		X		X	X	X		X			X	X				X		X	X	X		X		X	X	X	X	X			X	X	X	X		X	X		
145	X	X	X	X	X		X				X	X				X	X	X	X	X	X				X			X		X	X			X		X	X	X	X		
146	X	X	X		X	X	X		X	X		X			X	X	X	X	X	X	X		X		X	X	X	X		X	X			X	X				X	X	X
147	X	X			X	X	X		X	X		X		X	X	X		X						X	X		X			X				X		X		X	X		
148		X		X	X	X	X		X	X	X		X				X	X	X		X		X		X	X			X				X			X	X	X		X	
149	X	X	X	X	X	X	X	X		X	X	X	X	X		X				X		X	X	X			X			X		X	X	X		X		X	X	X	
150		X	X	X	X		X			X	X		X	X	X	X		X		X	X	X		X	X	X				X	X	X					X				X

For each value of counter: **DataID value to be used**

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
151	2	230	123	179	191	97	109	128	3	250	177	75	91	76	138	61
152	52	10	195	127	65	59	242	210	183	42	230	146	180	165	208	147
153	43	237	63	125	75	241	99	130	198	228	8	213	194	148	11	139
154	86	103	30	159	193	108	131	194	250	5	49	20	151	156	210	136
155	14	30	191	108	127	250	192	20	138	210	82	60	139	166	146	173
156	109	76	158	163	222	161	126	44	150	164	83	33	13	6	238	152
157	79	48	99	185	163	148	243	36	111	44	110	221	69	94	33	84
158	185	36	221	84	152	229	183	135	81	219	113	49	249	38	67	109
159	26	4	157	65	171	1	91	42	143	110	70	208	216	168	71	45
160	193	5	210	217	174	147	177	144	66	124	96	51	70	184	106	225
161	5	217	147	144	124	51	184	225	112	58	172	26	191	223	154	198
162	82	100	237	93	128	125	48	150	241	61	186	130	165	185	228	151
163	95	205	52	129	198	10	32	193	195	153	19	127	61	141	65	5
164	102	244	168	200	187	167	175	43	217	123	192	80	86	15	206	237
165	231	71	240	122	55	118	174	41	247	87	183	131	27	124	141	21
166	96	172	61	238	173	182	68	104	180	77	116	105	43	31	137	14
167	90	119	95	234	63	205	249	46	52	241	44	129	96	55	198	91
168	127	210	146	147	45	66	90	51	18	106	128	112	233	119	62	26
169	176	138	160	120	38	145	172	216	248	18	63	202	174	238	149	12
170	63	241	198	213	11	153	111	235	65	245	86	162	173	7	204	78
171	41	21	156	69	13	214	220	239	28	40	74	186	189	188	184	249
172	162	174	45	124	9	106	43	58	62	47	20	154	195	237	115	220
173	142	96	128	172	20	61	29	238	228	173	107	182	200	68	89	104
174	205	129	10	193	153	127	141	5	59	162	67	210	182	114	42	217
175	75	228	11	180	216	245	33	134	204	12	46	53	144	19	34	233
176	178	27	185	203	68	36	195	16	221	31	92	84	50	59	152	189
177	225	198	4	153	188	65	102	162	1	204	14	42	89	244	110	174
178	32	141	242	114	244	165	115	177	194	200	117	90	152	101	167	184
179	51	26	181	4	212	157	46	65	6	171	105	1	120	91	231	42
180	54	240	56	118	32	247	45	131	232	141	190	192	185	106	242	156
181	223	215	46	102	130	91	218	244	193	139	56	168	164	39	235	200
182	101	50	239	83	209	249	245	88	170	99	157	55	133	53	116	218
183	58	220	212	188	248	171	130	204	231	236	52	110	66	139	83	92
184	179	128	75	61	60	228	163	182	11	89	223	180	167	44	216	105
185	110	25	122	113	218	41	206	115	131	39	146	21	6	3	57	101
186	71	122	118	41	87	131	124	21	192	57	38	156	203	58	114	69
187	153	162	42	174	92	45	200	124	226	9	108	106	17	43	113	58
188	119	234	205	46	241	129	55	91	10	213	152	193	172	87	153	168
189	91	168	5	167	78	217	13	80	147	206	64	144	14	40	124	63
190	190	169	251	149	117	79	89	178	209	85	220	48	113	137	76	27
191	187	123	206	97	176	3	197	75	58	138	90	23	193	158	160	228
192	165	90	136	119	80	95	239	234	77	63	163	205	142	249	225	46
193	60	89	216	137	112	12	73	17	34	181	130	243	3	86	126	195
194	98	24	22	231	239	54	153	71	31	249	243	240	149	162	170	122
195	140	199	103	242	91	159	41	165	108	168	135	194	33	21	5	90
196	139	187	78	123	233	206	49	97	124	176	165	3	129	197	132	75
197	130	139	235	187	134	78	229	123	174	233	242	206	205	49	121	97
198	230	179	97	128	250	75	76	61	23	60	184	228	168	163	120	182
199	16	189	135	117	247	109	66	85	197	192	37	76	218	112	64	29
200	84	135	49	109	232	197	18	76	100	64	101	158	87	74	96	163

For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**

#	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
151		X	X				X	X	X			X	X							X	X		X	X		X		X	X		X	X		X	X	X	X	X	X	X	X
152					X		X		X			X	X	X	X		X	X	X					X	X				X	X		X	X			X		X	X	X	
153		X	X			X		X			X	X	X		X				X	X	X			X		X		X	X	X			X		X	X		X		X	X
154	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X			X		X			X	X	X		X	X	X		X			X	X	
155			X	X		X		X	X			X	X	X			X	X	X	X	X	X	X	X	X	X				X	X	X	X				X	X		X	X
156	X		X		X		X		X	X		X		X		X	X	X			X	X		X	X	X	X							X	X			X		X	
157	X	X	X		X		X	X	X					X	X		X	X	X		X	X			X	X	X	X			X	X			X	X	X	X		X	X
158	X	X						X	X	X	X	X	X	X	X					X	X	X		X	X				X			X	X	X	X	X	X	X	X	X	
159	X	X	X	X		X		X		X			X		X		X	X	X		X	X	X		X	X		X	X			X	X	X	X			X			X
160	X	X			X			X		X	X		X	X			X			X	X	X		X	X	X	X	X	X	X		X		X		X			X		X
161			X		X	X	X	X	X		X		X	X	X		X		X	X	X	X	X						X	X	X			X		X			X		X
162	X	X		X				X	X	X	X	X		X	X		X	X	X	X			X	X	X	X	X	X					X	X	X	X			X	X	X
163			X	X	X	X	X		X	X	X		X			X	X	X	X		X	X		X	X	X	X	X		X	X	X	X				X		X		
164	X			X		X				X		X	X		X	X	X		X	X	X		X	X	X	X	X	X			X	X	X	X			X	X		X	X
165	X		X		X						X	X	X	X	X	X			X			X	X			X	X				X	X	X	X	X	X		X			X
166		X	X			X	X			X		X				X		X	X				X		X	X			X	X		X	X		X		X	X	X		X
167				X	X	X	X	X				X		X	X			X		X		X	X	X			X		X	X	X	X	X	X	X	X	X		X	X	X
168	X	X		X	X	X	X		X	X		X			X	X	X		X	X	X	X		X	X	X		X	X	X				X		X			X	X	
169	X		X		X	X	X	X	X	X	X	X		X			X	X	X	X	X	X	X	X	X		X			X	X	X	X		X	X	X	X	X		X
170	X	X			X		X			X	X					X		X		X	X			X	X	X	X	X	X		X	X	X	X	X	X	X			X	X
171			X	X	X	X	X	X	X		X		X		X		X	X		X	X		X	X	X	X		X					X		X			X			X
172		X	X	X		X	X	X	X	X	X	X	X			X		X		X		X		X	X		X		X	X	X	X		X		X				X	
173	X	X		X		X	X	X	X	X	X		X	X			X		X	X	X	X			X	X			X	X	X			X	X	X		X			X
174	X	X	X	X	X		X		X	X	X	X	X	X		X	X	X	X	X	X	X		X		X	X	X	X	X	X	X	X	X	X		X	X			
175	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X	X	X	X				X	X		X	X	X	X	X		X		X	X	X	X	X	X	
176	X		X	X		X	X	X		X	X		X	X		X				X	X	X	X		X		X			X	X						X				X
177	X		X	X	X	X	X							X		X		X	X	X			X		X	X	X		X		X	X	X						X	X	X
178	X	X	X	X	X	X	X		X	X	X	X	X			X	X	X	X	X	X		X	X	X	X	X		X	X	X				X	X		X			X
179		X	X		X	X	X		X	X	X	X		X	X	X	X	X			X	X	X		X	X	X					X		X	X			X	X	X	
180		X		X	X		X		X		X	X	X	X		X	X	X			X	X	X	X		X					X	X	X		X	X	X		X		
181			X	X	X	X	X	X				X		X	X	X			X			X			X	X		X			X	X	X	X	X		X	X	X		X
182		X		X	X	X	X	X				X	X		X			X		X			X	X	X	X			X	X	X	X	X	X	X	X	X		X		
183	X	X			X			X		X	X	X		X					X	X	X		X	X	X	X				X	X	X		X	X	X		X	X	X	X
184	X		X	X	X			X	X	X		X	X	X			X	X	X		X	X			X	X	X	X		X	X		X					X	X	X	
185			X	X	X	X	X	X		X	X		X		X		X	X	X	X	X				X	X	X	X	X	X	X	X		X		X		X			X
186			X	X		X	X	X		X	X			X		X			X	X	X	X			X					X		X			X		X	X	X	X	
187			X	X	X			X	X	X	X			X	X	X	X	X				X				X					X	X	X		X	X	X			X	
188	X	X	X		X	X		X	X	X	X		X	X	X		X	X		X	X	X			X	X	X		X			X		X	X	X	X		X	X	X
189	X	X		X		X	X	X	X		X	X	X	X	X	X				X	X	X	X			X	X	X			X		X	X	X	X	X		X	X	X
190	X	X	X	X				X			X	X			X	X	X	X	X	X		X	X			X	X		X	X	X	X	X	X	X		X	X	X	X	X
191	X		X		X			X	X	X	X	X		X	X	X		X	X	X	X		X		X	X		X	X		X	X	X	X			X	X		X	
192		X					X		X		X	X		X	X	X	X	X	X		X		X	X	X	X			X	X	X					X		X	X	X	X
193	X						X	X	X		X			X	X		X	X	X		X	X	X		X		X	X	X	X	X		X	X	X	X		X		X	
194	X		X	X			X		X	X	X	X	X		X	X	X	X	X	X	X		X			X	X						X				X	X	X		X
195					X	X	X	X	X	X	X	X				X	X	X	X	X	X	X			X	X		X	X			X	X							X	X
196	X		X	X	X	X	X	X	X			X		X								X		X	X	X		X	X	X	X	X		X			X	X	X	X	
197	X		X	X		X		X		X		X				X	X					X		X			X		X			X	X	X	X				X	X	
198			X	X	X	X		X		X		X				X	X				X	X	X	X	X			X			X	X		X	X	X	X			X	X
199	X				X		X	X	X		X	X		X	X				X	X	X	X	X	X		X	X		X		X	X	X	X	X				X	X	
200	X	X	X	X	X	X					X	X		X	X	X	X	X	X	X	X	X			X					X					X	X	X			X	

#	For each value of counter: **DataID value to be used**																For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**																																								
#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
201	77	52	17	10	4	195	199	127	94	65	2	59	228	242	1	210			X		X		X	X	X	X		X	X		X	X	X		X		X	X	X	X	X	X		X	X	X				X	X		X	X		X	
202	111	7	244	81	2	200	72	82	167	230	131	43	140	251	123	100	X				X	X	X		X	X	X	X	X	X	X		X	X		X		X	X				X	X		X			X	X							X
203	228	180	245	134	12	53	19	233	92	243	91	121	225	67	207	176		X	X			X	X		X	X		X	X		X		X	X	X	X	X		X	X		X		X	X	X			X		X	X		X	X	X	X
204	227	22	68	54	186	31	65	240	152	170	207	56	79	42	140	118	X	X		X			X	X			X	X	X	X	X	X		X	X	X	X	X		X		X	X		X	X			X	X		X	X	X			X
205	238	104	105	14	52	70	56	30	233	10	7	191	125	247	195	108		X	X	X	X	X	X	X	X		X		X	X	X				X	X	X			X	X	X			X		X	X	X		X	X	X	X	X	X	
206	207	201	35	190	16	72	250	169	175	189	124	251	110	60	135	149	X	X	X		X	X		X	X	X		X	X	X	X		X	X		X	X	X		X	X	X	X	X		X		X		X	X	X	X	X		X	
207	40	107	223	116	93	215	196	111	46	150	54	102	166	246	130	7	X		X	X	X			X	X					X							X	X				X		X	X	X	X			X	X	X	X		X	X	X
208	133	227	29	22	214	68	4	54	44	186	34	31	251	65	8	240	X	X	X	X	X	X	X	X		X	X		X	X	X		X	X	X	X	X	X	X	X	X		X	X		X		X	X	X	X			X		X	X
209	97	75	23	228	120	11	161	180	188	216	234	245	217	33	202	134		X	X	X	X	X		X		X	X							X	X	X	X		X	X	X		X		X	X	X	X	X			X	X	X	X	X	X
210	57	13	177	40	82	184	248	107	119	100	227	223	247	236	237	116	X			X		X	X				X	X	X											X		X	X	X	X	X		X			X		X	X	X		X
211	236	196	88	246	185	218	233	35	87	36	42	39	24	176	221	72	X	X	X	X		X	X	X	X	X		X	X		X		X		X			X				X	X	X	X	X				X		X			X	X	X
212	191	250	138	60	146	120	222	89	145	66	237	216	78	164	18	137		X		X		X			X	X	X	X	X	X		X	X			X	X	X		X	X	X		X	X			X	X	X	X	X	X			X	
213	81	82	43	100	179	237	79	93	63	128	214	125	242	48	75	150		X	X		X	X		X	X	X				X		X	X				X		X			X	X	X	X		X	X	X	X		X				X	X
214	30	108	250	20	210	60	166	173	120	147	100	89	187	28	66	77	X	X	X	X	X		X	X	X	X	X	X	X		X	X		X	X	X	X	X		X	X			X	X		X	X	X	X		X		X	X	X	
215	99	148	111	221	33	7	201	229	244	19	122	81	186	190	2	49	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X							X	X	X		X				X	X	X	X	X		X		X		X
216	237	125	241	130	228	213	148	139	153	180	140	235	136	221	245	187	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X				X	X		X		X	X					X
217	244	200	167	43	123	80	15	237	144	97	166	63	103	209	3	125	X	X		X	X	X	X	X	X		X					X		X	X					X	X	X	X				X		X	X	X	X		X			
218	100	93	125	150	61	130	185	151	213	182	170	139	90	36	180	2				X	X			X	X	X			X		X	X		X	X		X	X				X	X		X	X		X	X		X			X		X	
219	164	73	104	86	205	14	240	103	70	129	221	30	93	118	10	159		X		X	X					X	X			X	X	X	X			X		X	X					X		X	X	X	X	X		X			X	X	X
220	221	229	81	49	67	82	169	197	43	142	21	100	32	149	179	158	X			X				X		X	X	X	X	X	X		X	X		X	X				X	X	X			X	X	X		X	X		X	X		X	
221	27	203	36	16	31	84	59	189	229	56	9	135	83	146	219	117		X	X		X	X		X	X				X		X		X		X				X	X			X	X			X	X	X	X		X	X				
222	13	40	184	107	100	223	236	116	234	93	22	215	192	196	125	111			X		X	X	X	X	X	X		X	X		X		X			X		X	X	X			X	X	X			X	X		X		X			X	X
223	177	184	119	223	237	234	83	215	205	125	68	46	64	88	241	102	X			X	X	X	X	X	X	X	X	X				X	X	X		X	X	X		X		X	X					X	X	X	X						X
224	34	207	246	201	203	35	191	190	39	16	174	72	171	250	84	169	X		X	X	X	X	X		X	X		X	X		X	X	X	X	X		X								X	X	X	X	X		X	X		X	X	X	
225	36	84	229	135	219	49	38	109	82	232	115	197	55	18	142	76	X	X		X	X	X		X	X	X	X	X	X	X		X		X	X	X		X		X	X			X	X	X	X			X		X	X				
226	198	153	65	162	204	42	244	174	208	92	30	45	137	200	25	124		X		X	X	X	X		X				X	X					X		X	X	X	X	X		X		X		X		X	X	X	X	X	X			X
227	103	159	108	194	5	20	156	136	60	217	197	173	2	214	147	95				X		X		X		X		X			X	X		X	X				X	X	X	X	X	X	X	X	X	X	X			X		X		X	X
228	222	164	238	73	95	104	54	86	105	205	148	14	100	240	52	103	X	X		X	X	X	X	X			X	X	X			X		X	X	X	X			X	X		X		X		X	X	X	X	X		X		X		X
229	65	42	208	45	25	226	167	106	224	113	250	62	243	80	41	154	X	X		X	X	X			X			X			X	X	X	X	X				X		X	X	X	X	X	X	X				X	X	X	X	X	X	X
230	167	80	144	63	3	225	107	241	26	23	164	198	108	116	220	213	X			X	X	X	X	X	X		X	X	X	X	X		X		X		X	X	X	X	X		X	X	X	X	X	X		X		X	X		X	X	X
231	48	185	148	36	44	221	94	84	7	152	25	229	239	183	19	135	X	X	X	X	X		X		X		X	X		X	X	X	X		X	X	X	X		X	X				X	X		X	X		X	X		X		X	
232	66	112	74	181	98	126	205	157	27	24	180	6	160	129	22	1	X	X	X			X		X	X	X		X			X	X	X	X	X		X					X	X	X	X	X	X	X	X	X		X			X	X	X
233	59	146	38	66	226	18	136	112	149	62	75	74	121	95	133	181	X	X		X	X		X	X	X		X		X	X		X	X	X	X		X	X		X			X		X			X	X		X	X	X	X	X		X
234	243	94	201	183	143	190	108	38	72	211	206	169	204	20	189	18	X		X		X			X	X	X		X	X	X	X	X	X		X	X		X		X			X					X		X	X	X					
235	150	151	139	2	105	187	84	230	78	70	141	123	234	135	233	179	X	X		X		X	X		X	X				X	X		X	X	X		X	X			X	X	X			X	X		X	X	X			X		X	X
236	220	188	171	204	236	110	139	92	71	196	10	25	112	187	88	9	X	X			X	X							X	X	X	X	X	X			X	X		X			X	X	X	X	X	X			X	X	X	X	X	X	X
237	210	147	66	51	106	112	119	26	74	154	61	181	176	234	98	4	X	X	X		X		X			X	X							X	X	X		X	X				X	X		X				X		X	X		X	X	
238	235	78	174	206	121	124	82	3	106	132	194	58	10	100	47	23		X		X		X			X	X					X				X	X	X	X	X	X		X		X	X	X	X	X	X	X	X	X	X	X		X	X
239	143	211	189	224	118	117	147	133	109	131	160	85	246	51	192	227		X			X		X	X				X			X		X						X			X			X	X	X			X	X		X	X		X	X
240	134	233	121	176	94	132	142	138	47	183	167	160	153	96	190	120	X	X		X	X	X			X		X		X		X	X	X	X	X	X			X				X	X			X	X	X			X	X	X			X
241	74	126	27	6	22	203	10	143	36	54	53	16	248	127	31	211	X	X	X				X	X		X	X	X	X		X	X	X	X		X		X	X	X			X			X	X	X		X	X					X	X
242	181	157	6	1	231	143	193	208	16	71	233	211	202	5	240	226	X	X							X	X	X	X	X				X	X	X		X		X	X	X	X		X	X	X		X	X	X	X	X	X		X	X	X
243	61	182	180	105	137	134	152	70	53	17	102	233	63	219	243	191	X	X		X	X			X		X	X	X	X		X			X		X				X			X	X	X	X		X			X	X	X	X	X	X	
244	83	88	55	218	148	87	121	39	141	221	208	57	22	132	7	175		X	X	X		X	X	X		X		X		X	X	X	X	X	X	X	X	X	X		X		X		X	X	X	X			X		X		X	X	
245	195	59	183	146	208	38	194	66	169	226	97	18	53	136	224	112		X				X	X	X		X		X			X		X	X	X	X	X			X	X	X			X	X			X	X	X	X		X	X		X
246	64	222	172	164	136	238	22	73	182	95	99	104	82	54	77	86	X	X	X	X			X	X	X	X	X		X	X				X	X		X	X	X		X	X	X		X	X	X					X	X		X	X	
247	196	246	218	35	36	39	176	72	57	84	45	175	231	138	229	251	X						X	X	X	X	X			X	X	X		X	X		X	X	X	X	X	X	X				X	X	X	X	X		X	X	X		X
248	25	113	41	115	39	21	3	101	156	175	66	69	143	23	13	50	X	X	X	X	X		X	X	X		X	X		X	X				X	X	X	X	X		X	X	X	X		X	X			X	X	X			X	X	
249	226	62	133	98	21	227	225	24	29	69	216	22	190	198	214	231			X	X		X	X	X	X					X	X	X		X	X			X		X		X		X	X	X	X	X	X		X		X	X		X	X
250	239	249	170	55	116	32	92	87	199	111	143	141	29	9	102	57		X	X	X	X	X	X		X	X		X		X		X	X	X	X		X	X			X	X		X	X		X	X		X			X	X	X		

For each value of counter: **DataID value to be used** | For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
251	50	187	202	251	61	42	248	144	169	101	145	128	37	190	160	47	X						X			X	X	X		X	X	X	X	X	X	X	X			X		X	X	X	X	X		X	X	X	X				X		
252	24	10	208	90	84	149	222	154	114	192	118	228	245	12	19	92	X	X		X			X	X	X		X	X				X	X	X	X	X	X	X		X	X	X	X		X	X		X	X	X	X	X	X	X		X	
253	138	137	193	104	228	96	88	41	119	6	38	175	161	166	117	162	X				X	X		X	X				X	X		X	X			X	X	X	X				X	X	X	X	X			X	X		X		X	X	
254	37	71	91	229	237	35	20	186	58	17	9	200	174	153	76	222			X	X	X		X			X	X		X	X	X	X			X	X	X	X	X		X			X	X	X	X		X	X	X		X	X	X		X
255	120	125	84	196	205	192	128	215	19	22	83	93	162	234	113	116			X	X			X			X	X		X	X				X		X	X		X			X		X		X	X	X	X						X	X	X
256	144	47	234	158	118	40	77	127	1	230	63	38	186	25	163	51	X	X	X		X		X	X		X		X	X		X	X	X	X					X	X		X	X	X			X	X	X	X	X	X	X		X	X	X
257	164	84	214	192	180	19	121	93	168	113	138	236	179	227	153	107	X	X	X	X	X	X					X	X	X	X		X		X	X	X			X	X		X	X		X			X	X		X	X		X	X	X	X
258	156	207	150	221	37	102	206	33	246	71	232	7	238	214	91	201	X			X		X	X	X	X		X		X		X		X	X		X		X	X	X		X	X			X	X	X		X	X		X				X
259	184	208	212	149	214	114	131	228	134	19	243	225	176	15	168	6		X		X	X		X	X					X	X		X	X		X		X		X	X	X	X	X	X		X	X		X	X		X	X	X		X	X
260	176	78	22	34	152	223	118	209	13	52	199	59	164	230	190	147	X		X	X	X	X		X	X		X		X	X	X	X	X		X	X		X			X	X	X	X	X			X		X				X	X	X	
261	216	142	86	58	24	204	201	83	21	10	144	138	62	122	208	100		X			X	X	X		X		X		X	X	X		X	X	X		X	X	X		X	X		X			X	X		X	X		X			X	X
262	234	40	1	38	163	53	3	62	249	79	37	30	131	49	210	126			X	X				X	X	X	X	X	X	X	X			X	X	X	X	X			X	X		X	X			X		X			X	X	X	X	
263	240	168	105	82	73	172	92	123	115	213	147	156	248	55	23	44	X	X	X	X					X	X		X	X	X			X		X	X	X	X	X	X	X		X					X	X	X		X		X	X		
264	49	29	251	57	246	144	52	211	128	151	153	47	94	134	28	171	X	X			X	X	X			X	X	X	X	X			X	X		X	X	X	X	X	X	X	X				X	X	X	X		X	X	X		X	X
265	197	165	97	233	187	133	22	4	186	251	240	108	207	223	42	241		X	X	X	X	X			X		X	X		X		X	X			X			X		X	X			X			X		X	X		X	X	X		X
266	226	95	110	157	197	69	125	20	33	165	103	145	182	196	97	153	X	X	X	X	X		X				X	X		X	X	X	X	X		X	X		X		X			X	X	X	X		X	X	X	X	X			X	
267	87	32	55	249	155	250	60	120	89	66	216	164	137	202	112	125	X			X	X		X				X	X	X	X	X	X							X	X		X	X	X	X			X		X	X	X	X	X	X		X
268	6	239	64	89	226	67	56	206	220	95	245	130	147	76	110	214	X	X			X	X	X		X	X	X	X	X	X	X					X	X	X	X	X	X	X			X	X				X	X	X	X	X	X	X	
269	34	209	59	147	134	18	188	119	75	31	32	181	192	235	77	159		X		X	X	X			X				X	X	X			X			X	X	X		X	X	X					X	X	X		X		X	X	X	
270	202	42	169	128	160	121	8	234	199	36	122	227	91	43	161	40	X	X			X	X	X	X	X	X	X		X			X	X	X	X	X		X					X		X		X		X		X		X	X	X		
271	20	153	73	11	4	213	116	198	23	241	225	63	80	155	203	16	X	X	X	X		X		X	X	X	X	X		X		X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X				X
272	41	162	179	184	89	103	35	208	152	206	160	212	210	200	130	149						X		X		X	X	X		X		X	X	X	X	X	X	X	X		X	X		X	X	X							X		X	X	X
273	38	62	30	126	177	219	29	160	125	245	229	231	45	57	176	36			X	X	X		X	X	X		X	X			X	X		X	X	X		X	X	X			X	X	X	X	X	X	X		X		X	X		X	X
274	209	147	18	119	31	181	235	159	132	182	249	27	93	65	25	224	X			X	X	X	X	X	X	X		X	X		X	X	X	X	X		X		X		X	X			X	X	X	X	X		X		X	X		X	X
275	98	180	20	113	124	153	161	13	73	139	149	11	67	148	4	194	X	X			X		X	X	X				X		X				X						X	X	X			X	X			X		X		X		X	X
276	105	172	115	156	23	46	239	207	174	189	181	150	8	89	88	221	X		X	X	X	X			X		X	X	X		X				X									X	X	X	X	X	X	X	X		X	X	X	X	
277	171	48	127	174	140	51	39	54	62	191	197	61	203	220	235	246		X						X		X	X	X	X	X	X	X			X	X			X	X	X	X			X	X				X	X	X	X	X		X	
278	33	201	244	122	186	2	236	23	72	222	14	167	29	195	131	189	X		X		X			X		X		X	X	X	X			X		X		X	X	X	X		X	X		X	X						X	X	X	X	X
279	22	223	13	59	190	194	163	18	198	14	92	75	214	79	43	181	X	X	X	X			X	X	X		X	X	X	X	X	X		X	X	X		X		X			X	X	X		X	X	X		X	X	X	X		X	
280	80	109	242	105	201	68	228	172	171	122	59	115	124	6	2	156	X			X	X	X	X		X	X		X		X	X	X		X	X		X	X	X		X	X	X	X	X	X	X	X	X				X	X	X		X
281	150	102	246	7	91	151	240	244	28	35	139	81	50	168	58	2			X	X	X		X			X	X	X	X				X		X				X	X		X		X		X	X	X	X	X	X					X	X
282	47	158	40	127	230	38	25	51	53	188	247	62	222	143	79	61	X	X	X	X	X	X	X	X		X	X			X	X	X	X	X	X	X	X	X	X	X	X					X	X	X		X		X		X	X	X	X
283	117	64	39	67	44	220	24	130	141	110	176	80	18	10	221	240	X			X	X	X	X	X	X	X			X	X		X	X	X	X	X		X	X	X	X		X	X	X	X		X					X	X	X		
284	108	203	12	85	234	5	75	170	45	40	156	70	244	132	1	60					X		X		X		X	X	X	X				X				X	X						X		X	X	X	X			X		X		X
285	229	186	200	222	169	131	11	12	100	121	68	15	246	198	199	5		X		X	X	X		X			X			X	X			X				X		X	X	X	X	X	X			X	X		X		X	X		X	
286	228	6	117	239	195	64	26	89	39	226	177	67	209	56	44	206	X	X			X		X	X		X		X	X	X	X			X	X		X	X	X		X		X	X	X	X	X	X		X	X	X	X	X	X	X	
287	17	169	217	121	231	199	241	227	154	161	2	87	142	135	215	1	X	X	X	X				X	X	X	X	X		X		X	X	X	X	X	X	X	X	X		X	X			X	X	X		X	X	X	X	X		X	X
288	172	156	46	207	189	150	89	221	54	37	27	102	43	206	166	33	X		X		X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X	X	X	X		X	X	X				X	X	X		X	X	X	X	
289	168	82	172	123	213	156	55	44	46	63	119	207	190	250	189	110	X				X		X	X	X	X		X			X	X	X	X		X				X	X	X			X	X	X	X			X			X		X	X
290	188	235	177	65	41	245	102	86	176	162	202	173	1	7	179	204	X			X		X	X		X	X	X	X			X						X			X	X	X		X	X			X		X		X	X	X	X		
291	56	76	10	217	125	90	108	199	149	196	81	154	65	203	192	87	X	X				X		X	X		X	X	X	X			X	X	X	X	X			X		X				X		X		X	X	X	X	X	X		
292	147	119	181	159	182	27	65	224	129	39	120	232	107	86	143	152	X				X	X	X	X				X	X		X	X			X	X		X		X	X	X	X	X			X	X	X	X	X	X	X	X	X		X
293	89	206	130	214	157	240	217	19	109	20	21	168	159	199	145	236	X		X	X		X	X	X	X	X	X	X	X		X		X			X	X		X		X	X	X	X		X		X	X		X	X	X	X			X
294	94	97	229	133	202	186	113	108	200	42	109	222	150	13	169	203			X	X	X	X	X	X	X				X	X	X		X		X		X	X				X	X	X	X	X	X		X					X	X		
295	186	222	131	12	121	15	198	5	183	227	115	45	151	16	87	70					X	X	X	X		X	X	X	X	X	X	X	X	X		X		X		X	X			X	X		X			X			X			X	
296	66	74	98	205	27	180	160	22	20	232	10	113	239	36	124	223	X	X	X	X	X	X		X	X			X		X	X		X	X	X		X	X	X	X	X	X	X	X		X			X	X	X	X	X		X	X	X
297	8	43	135	238	171	218	119	3	170	48	226	50	73	159	127	49		X					X	X	X						X				X		X		X	X	X	X			X		X		X		X	X	X		X	X	X
298	241	135	85	218	158	170	181	50	70	127	44	60	122	27	38	187				X	X			X	X		X		X		X	X	X		X			X		X	X		X	X		X	X	X	X	X	X		X	X	X	X	X
299	39	220	141	80	221	9	208	109	57	33	22	242	132	149	7	105	X	X				X				X	X	X	X	X	X	X	X	X	X	X	X			X	X	X			X	X	X			X	X			X	X	X	
300	217	199	154	87	215	92	85	32	6	116	140	55	204	170	107	249		X	X				X	X			X		X		X			X	X	X	X		X	X	X	X	X	X	X	X		X	X		X	X	X	X	X		X

#	For each value of counter: **DataID value to be used**																For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**																																							
#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
301	123	44	207	110	247	221	164	69	102	94	224	33	77	84	71	145		X	X	X		X		X		X			X	X	X	X	X					X	X				X	X		X	X		X		X		X	X	X		
302	72	140	188	191	183	235	207	216	177	136	50	65	234	221	41	142	X			X		X	X		X	X		X	X	X					X		X		X	X	X		X		X		X		X		X		X	X		X	X
303	179	103	152	212	130	52	83	114	190	240	161	134	74	100	109	225	X	X	X	X	X	X		X	X	X	X	X		X	X	X	X				X	X		X	X	X	X		X		X	X	X	X	X				X	X	X
304	242	68	171	115	2	48	117	46	127	167	75	174	4	64	140	150	X		X	X	X		X	X	X	X	X	X		X	X		X		X		X		X	X		X	X	X									X	X	X	X	X
305	190	14	43	77	68	238	96	25	218	115	146	3	153	175	48	143	X			X		X	X	X				X	X		X			X		X	X			X		X			X		X	X			X		X			X	
306	208	149	114	228	19	225	15	6	31	236	163	117	78	45	82	239		X	X			X	X	X	X	X	X	X	X				X	X		X	X			X	X	X					X			X		X	X			X	
307	149	228	225	6	236	117	45	239	182	195	79	64	34	26	123	89		X	X	X	X	X		X	X				X	X		X	X		X		X		X	X	X	X	X	X					X		X	X	X	X	X		X
308	103	212	52	114	240	134	100	225	14	168	148	31	205	183	105	117	X	X			X	X	X	X	X	X		X		X			X	X	X		X				X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X
309	249	120	164	125	74	84	42	196	214	205	58	192	41	128	180	215	X		X	X	X	X		X	X	X		X	X	X	X			X	X			X		X		X		X		X			X		X		X	X			
310	185	193	147	96	225	119	191	175	181	117	53	159	215	216	182	179	X	X	X			X	X	X		X	X	X	X	X	X	X			X	X	X		X	X		X	X	X	X		X	X	X			X	X	X	X		
311	205	22	113	223	248	13	243	59	11	190	154	194	206	163	8	18	X		X	X	X		X	X	X	X	X			X		X	X	X	X	X	X		X	X	X	X	X		X			X		X			X	X	X	X	X
312	79	177	106	245	175	176	246	173	205	179	17	78	32	151	224	21	X	X	X	X	X					X		X	X	X	X				X			X	X	X	X	X	X	X		X	X	X		X	X			X	X	X	X
313	198	16	247	197	218	94	74	165	71	50	67	97	172	205	60	233	X	X	X		X	X	X		X	X	X		X		X	X	X	X	X	X	X		X		X		X		X				X	X	X	X	X	X	X		X
314	116	155	146	66	18	112	62	74	95	181	178	98	228	126	132	205		X	X		X	X	X	X		X	X	X	X	X		X	X		X	X			X	X		X					X	X	X		X	X	X	X			
315	31	182	25	39	156	143	162	220	49	207	74	141	194	184	150	80	X				X	X	X	X			X	X	X		X	X	X	X		X	X		X	X		X					X	X	X	X	X						
316	104	41	175	162	239	179	91	184	224	89	126	103	99	35	67	208	X		X			X	X	X	X	X		X		X		X		X			X	X	X		X	X		X	X	X	X	X			X				X		
317	27	232	124	248	141	139	21	190	4	9	192	8	95	185	57	14		X	X	X	X		X	X		X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X	X	X		X	X		X		X	X	X		X	X
318	70	237	56	17	30	76	233	169	10	219	7	217	191	4	125	121	X		X			X	X	X	X		X	X		X	X	X	X		X	X		X			X	X	X	X	X	X		X	X	X			X			X	X
319	162	184	103	208	206	212	200	149	52	214	36	114	106	131	240	228	X	X	X					X			X			X		X		X	X	X		X	X	X	X		X	X	X	X	X	X	X	X	X			X	X	X	X
320	189	37	166	71	70	91	157	229	142	237	141	35	48	20	56	186			X	X		X	X	X	X	X	X	X	X			X			X	X	X	X		X					X	X	X	X			X					X	
321	86	204	21	138	208	185	2	137	209	149	234	193	219	167	114	104	X	X	X	X	X	X	X		X		X	X			X	X			X	X	X		X	X		X		X	X			X	X	X		X	X	X		X	X
322	43	238	218	3	48	50	159	49	60	174	95	187	213	224	51	29		X		X	X	X	X					X	X	X		X		X		X		X	X	X				X			X	X	X		X	X	X		X		X
323	159	224	232	152	220	248	204	52	139	80	196	190	112	138	9	134		X	X	X	X	X					X							X	X	X	X				X	X	X	X		X		X	X		X	X		X	X	X	X
324	114	225	31	117	82	182	136	64	25	123	210	39	223	178	156	67		X	X		X	X			X	X		X	X			X	X	X	X	X		X		X	X					X		X	X	X		X		X	X	X	
325	53	30	120	219	106	125	251	231	84	176	35	196	136	144	205	161	X			X	X			X		X	X	X	X	X	X			X	X	X		X		X		X	X	X	X		X	X	X	X	X			X	X	X	X	X
326	135	218	170	50	127	60	27	187	237	51	110	202	23	232	62	251	X	X	X		X	X	X	X	X	X	X	X	X		X			X	X	X	X			X	X		X	X	X				X	X	X		X	X	X	X	X
327	102	7	151	244	35	81	168	2	118	200	8	72	187	82	83	167	X	X	X		X	X	X	X		X		X	X	X	X	X	X	X	X	X	X	X								X		X	X		X			X	X	X	
328	100	183	104	136	92	41	37	178	175	55	51	162	243	71	239	24			X			X	X	X	X	X	X	X				X			X	X	X		X	X				X	X	X	X	X	X					X		X	
329	101	28	36	118	204	243	68	230	148	138	203	163	17	115	185	188			X	X	X	X		X		X	X		X	X	X			X	X	X		X			X		X	X	X	X	X		X	X	X			X		X	X
330	35	200	83	131	217	100	213	15	137	199	171	183	101	63	154	45	X	X			X	X			X	X	X	X	X	X	X		X		X		X	X		X			X		X	X	X			X	X	X				X	
331	122	23	167	189	12	88	226	37	191	5	25	166	211	95	45	71	X		X	X	X	X	X	X		X	X	X	X	X	X			X		X		X	X		X		X	X				X	X	X		X	X	X	X	X	
332	77	25	3	143	46	49	179	141	187	150	98	29	198	103	54	9		X	X		X	X		X			X		X		X	X	X		X	X	X		X	X		X		X		X		X	X	X		X	X	X	X		
333	225	117	182	64	123	39	178	67	143	44	106	220	59	24	207	130			X		X					X		X	X	X	X	X	X			X	X	X	X	X	X			X		X	X	X	X			X			X		X
334	45	26	178	56	249	24	97	76	184	120	246	10	188	133	164	217		X	X						X				X		X		X					X	X	X	X		X	X		X	X			X		X	X	X	X	X	X
335	247	94	71	97	60	229	180	133	35	202	80	186	46	113	17	108	X	X	X		X	X	X	X	X	X	X	X						X					X		X				X		X		X	X	X	X	X		X	X	
336	139	8	241	43	211	135	147	238	85	171	195	218	145	119	158	3	X	X		X		X	X	X	X			X	X	X			X	X							X	X		X		X	X	X	X	X	X	X	X		X		
337	93	107	195	146	194	226	53	112	44	75	41	95	114	30	16	98			X			X	X	X	X		X	X	X	X				X		X		X		X	X	X		X	X	X		X		X			X	X			
338	195	226	44	95	16	110	120	157	221	197	179	69	31	125	94	20		X	X			X	X	X	X	X	X	X	X	X	X			X			X	X		X		X			X	X	X	X				X	X	X			
339	64	67	220	130	110	80	10	240	9	69	78	109	181	90	33	168	X	X				X	X		X	X			X		X	X	X	X			X	X		X		X	X	X	X					X	X	X	X		X		
340	222	12	15	5	227	45	16	70	136	1	46	26	81	197	32	237		X	X	X		X			X		X	X	X	X				X	X		X		X	X	X	X	X	X		X	X	X		X	X	X	X	X		X	
341	9	242	211	68	244	171	225	115	158	2	194	48	233	117	72	46		X		X	X	X		X		X	X			X	X			X	X	X	X	X	X		X	X	X		X	X	X		X	X	X	X	X			X	
342	207	221	102	33	71	7	214	201	151	229	248	244	3	19	35	122					X		X			X	X	X			X	X		X	X	X	X		X			X			X	X			X			X	X		X	X	X
343	23	189	88	37	5	166	95	71	216	70	143	91	171	157	26	229			X	X	X	X		X					X	X	X		X	X	X				X	X	X		X	X	X	X	X		X					X			X
344	166	91	142	35	56	58	145	200	204	76	57	83	51	73	10	131			X	X	X	X	X		X			X	X	X		X	X	X		X	X	X		X		X	X	X	X				X	X	X				X		
345	92	55	239	250	146	89	237	164	67	112	65	206	193	17	95	84		X	X	X	X	X			X	X	X	X	X	X		X	X	X		X		X		X		X	X		X		X	X	X	X			X				X
346	11	198	63	16	135	247	66	197	37	218	64	94	105	74	170	165	X	X			X		X		X					X	X			X		X	X	X		X				X	X	X			X	X		X	X	X	X	X	
347	194	75	16	132	238	197	106	129	94	3	89	165	82	176	50	124		X	X		X			X		X	X		X	X	X	X	X		X		X		X	X		X		X	X		X	X		X	X		X	X			X
348	129	124	233	139	29	4	34	8	108	57	19	241	110	209	144	43			X					X					X		X	X	X	X		X	X	X		X		X	X		X	X		X	X	X				X	X	X	X
349	180	113	153	13	139	11	148	194	213	8	228	198	130	99	241	75		X	X				X	X			X	X	X	X	X		X		X	X	X	X	X	X		X		X		X								X		X	X
350	223	59	194	18	14	75	79	181	16	77	55	132	19	177	238	27		X	X			X	X	X	X			X	X			X	X	X	X	X	X		X		X					X		X			X		X	X			X

	For each value of counter: **DataID value to be used**																For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**																																								
#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
351	109	105	68	172	122	115	6	156	48	23	18	46	139	239	167	207								X		X		X	X		X	X			X				X	X	X	X	X	X			X	X		X	X	X	X		X	X	X
352	187	251	42	144	101	128	190	47	121	28	73	234	71	14	36	158				X	X	X		X	X		X	X	X	X			X	X		X		X	X					X	X	X	X		X		X				X	X	
353	233	4	108	241	144	203	59	135	12	47	82	85	33	18	234	218	X			X	X	X		X		X	X	X			X	X	X		X	X	X	X	X	X	X		X	X	X	X			X	X	X	X	X	X	X	X	
354	174	54	61	246	216	101	80	151	160	142	233	28	170	109	86	81	X	X			X	X	X	X	X	X		X	X	X	X	X		X				X	X		X		X	X	X				X	X		X			X		X
355	213	63	189	247	85	37	112	94	166	170	39	71	68	98	70	97		X		X	X	X	X	X	X	X	X	X	X	X		X	X	X	X			X	X	X	X	X		X	X				X	X	X	X	X	X	X		X
356	18	181	132	27	25	129	245	232	165	143	164	124	195	173	49	248	X		X	X	X	X		X	X			X		X	X	X	X		X				X	X	X		X	X	X	X	X	X					X	X		X	
357	97	133	186	108	42	222	13	203	131	128	105	12	102	194	121	85	X	X			X	X	X		X		X	X	X		X	X		X		X	X	X		X	X		X	X	X	X	X	X	X		X	X		X	X	X	X
358	76	217	90	199	196	154	203	87	228	215	72	92	86	85	93	32	X	X		X	X	X		X		X	X		X	X	X		X			X	X	X		X		X	X	X	X		X								X	X	X
359	193	96	119	175	117	159	216	179	27	64	30	224	116	142	39	103	X	X	X		X	X	X	X		X			X	X			X		X		X	X	X		X	X		X	X	X		X	X	X	X	X	X	X	X		X
360	246	151	28	81	58	118	105	72	243	83	241	230	202	172	138	140		X		X	X	X				X	X	X	X	X	X		X	X	X		X	X	X	X	X		X	X					X		X		X	X		X	
361	212	114	134	225	168	31	183	117	77	82	99	182	22	136	172	64	X		X	X			X	X	X		X	X	X	X	X	X		X	X	X		X	X	X	X			X				X		X		X	X	X	X	X	X
362	16	197	94	165	50	97	205	233	229	187	130	133	156	22	202	4		X		X	X		X	X	X	X	X	X	X			X	X					X	X		X				X		X	X		X		X	X		X	X	X
363	141	9	57	242	7	211	114	68	47	244	13	171	165	225	81	115	X	X	X	X	X	X			X		X	X	X	X	X	X		X		X	X	X		X	X	X	X			X						X	X		X		X
364	125	196	192	215	22	93	234	116	236	223	100	107	184	40	13	155	X	X	X	X	X	X			X	X		X		X		X	X					X	X			X				X	X	X	X		X	X	X			X	
365	107	146	226	112	75	95	30	98	110	132	162	157	225	219	197	180	X			X	X	X		X			X		X			X		X	X				X		X		X		X	X			X	X			X	X			X
366	25	143	49	141	150	29	103	9	251	102	180	57	16	212	246	242		X	X	X	X	X			X	X	X						X	X	X	X	X	X	X	X		X		X	X		X	X	X	X	X			X	X		
367	90	154	228	92	93	6	5	55	117	107	188	239	21	70	195	250	X					X		X	X	X	X	X	X	X	X	X	X			X	X	X	X	X					X		X		X	X	X		X		X	X	X
368	4	241	203	135	47	85	18	218	5	158	123	170	201	181	40	50		X	X	X		X	X		X		X		X	X			X		X		X	X	X	X	X	X	X		X	X			X	X	X		X	X	X		
369	118	230	163	188	137	79	46	235	210	104	170	177	121	150	96	65		X	X	X		X			X			X	X	X	X	X	X	X	X		X	X	X		X	X		X	X		X				X		X	X	X	X	X
370	244	2	72	167	131	140	123	88	188	15	238	191	144	44	183	166	X		X			X			X	X		X	X	X	X	X	X	X	X	X		X	X		X		X					X	X		X	X		X	X		X
371	134	31	77	182	172	25	41	39	3	156	66	143	13	162	46	220		X	X	X	X				X	X			X		X	X		X	X	X		X	X		X			X		X			X	X	X	X		X	X		X
372	29	57	144	211	151	47	134	171	234	81	11	158	97	31	118	48				X			X	X	X	X		X		X		X		X		X	X	X	X		X			X	X	X	X	X	X	X	X				X		X
373	227	1	32	53	99	249	50	30	250	210	166	120	100	187	66	219		X	X		X	X	X	X			X		X				X	X	X	X	X	X	X	X	X					X	X	X			X	X			X	X	X
374	192	93	236	107	13	195	1	146	123	194	104	226	212	53	198	112		X		X		X		X	X		X		X	X	X					X	X	X	X			X	X		X	X	X	X	X		X	X		X		X	X
375	40	38	53	62	79	30	49	126	120	177	71	219	15	29	106	160			X		X	X	X	X	X	X	X	X			X	X	X	X	X	X	X			X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X		X
376	163	79	210	177	96	106	54	245	74	175	237	176	87	246	159	173			X		X	X	X	X	X	X	X			X	X	X	X	X	X	X		X		X	X		X	X	X		X			X	X				X	X	X
377	250	164	206	84	98	214	169	192	240	180	204	19	175	121	20	93								X	X	X	X		X	X		X	X	X	X	X	X	X			X	X	X	X	X	X	X		X	X	X		X		X	X	X
378	224	152	248	52	80	190	138	134	8	109	215	14	98	137	242	31	X	X	X		X		X	X	X	X	X	X	X		X	X	X	X	X	X	X		X				X	X		X				X		X		X	X	X	
379	13	194	198	75	43	16	210	132	247	238	239	197	168	106	218	129		X				X	X	X		X			X	X	X				X	X	X	X	X	X		X			X		X	X		X		X	X	X	X	X	
380	211	171	158	48	72	127	182	174	38	140	16	51	108	39	188	54	X	X	X		X		X			X	X	X			X	X		X	X		X	X		X	X	X	X							X		X	X	X	X		
381	1	53	249	30	210	120	187	219	164	106	91	125	183	251	74	231		X	X	X	X		X	X	X	X	X			X			X					X	X	X		X	X		X			X	X	X	X	X		X	X	X	X
382	137	104	96	41	6	175	166	162	159	239	62	179	148	91	64	184	X	X	X	X	X	X	X	X		X		X		X	X	X	X				X	X	X			X		X		X		X			X	X	X	X	X	X	X
383	160	36	161	243	21	148	171	163	116	185	12	99	76	48	209	79	X	X	X			X			X	X			X	X		X		X				X			X		X	X	X							X	X		X	X	X
384	19	236	82	195	11	123	32	226	156	198	96	44	52	249	63	95			X	X			X		X	X	X	X	X	X		X	X				X	X			X		X	X	X	X		X	X	X	X		X		X	X	X
385	99	210	66	106	119	74	61	176	98	159	56	205	92	101	27	78	X		X		X	X		X	X	X	X		X			X		X	X		X		X	X	X	X	X		X	X	X	X	X						X	X	X	X
386	146	112	95	98	132	157	219	180	69	129	184	20	117	231	165	113		X	X	X	X	X	X	X	X		X	X		X			X	X		X		X		X	X				X	X	X	X				X				X	
387	71	229	35	186	17	200	153	222	83	169	242	131	54	11	217	12				X		X		X		X	X	X		X	X	X	X		X	X	X		X	X	X	X		X	X	X	X	X		X	X		X		X	X	
388	55	250	89	164	112	206	17	84	130	98	86	214	96	169	157	192	X	X	X		X	X		X	X			X		X	X	X		X	X	X		X			X		X	X				X			X			X		X	
389	203	85	5	170	40	70	132	60	26	38	207	237	2	129	53	202	X	X		X	X	X		X	X			X	X			X	X	X	X		X			X				X	X	X		X	X	X	X	X	X	X		X	X
390	165	233	133	4	251	108	223	241	222	144	168	203	221	59	128	135	X							X	X	X		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X						X		X
391	3	49	187	29	54	251	152	57	42	246	20	144	247	52	101	211				X	X	X	X	X			X		X		X	X	X		X	X				X	X	X			X		X	X	X		X		X	X	X	X	
392	218	50	60	187	51	202	232	251	17	61	69	42	189	248	126	144	X	X			X		X			X				X						X	X	X				X	X			X	X				X		X	X	X	X	
393	238	3	50	49	174	187	224	29	202	54	157	251	63	152	61	57				X			X	X	X	X			X		X	X	X	X			X	X	X		X	X						X	X	X	X		X		X	X	X
394	82	123	156	44	63	207	250	110	150	247	159	221	14	164	37	69	X					X	X	X	X	X	X	X	X						X				X		X	X	X	X	X	X		X	X	X				X	X	X	X
395	133	108	222	203	128	12	194	85	15	234	172	5	7	75	227	170	X	X	X	X	X	X	X			X		X		X	X	X	X		X			X			X		X	X	X				X		X	X	X		X	
396	65	86	173	204	184	21	244	138	34	208	128	185	30	2	212	137		X	X	X	X	X		X	X	X	X	X	X	X	X	X		X	X	X	X	X	X				X	X	X	X		X	X		X	X	X	X		X	X	
397	239	89	67	206	95	130	76	214	80	157	173	240	119	217	69	19	X	X			X		X			X	X	X	X			X	X		X	X	X		X		X		X		X	X	X		X		X				X		X
398	61	101	160	28	86	36	242	118	161	204	108	243	237	68	21	230	X		X	X			X		X	X		X		X	X	X		X	X		X	X			X	X	X				X	X	X		X	X	X	X	X	X	X
399	119	159	27	224	39	232	86	152	124	220	125	248	146	204	141	52	X	X	X	X	X	X	X		X		X		X	X			X	X			X	X			X	X	X	X		X		X			X	X	X	X	X		
400	231	161	215	148	34	116	158	99	107	209	15	155	10	127	59	210	X	X	X	X	X		X	X	X	X		X	X	X	X	X	X	X			X	X			X	X	X	X	X	X	X	X	X	X	X					X	

	For each value of counter: **DataID value to be used**																For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**																																								
#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
401	200	131	100	15	199	183	63	45	104	87	48	136	28	247	92	26	X	X						X	X	X		X		X	X	X					X	X	X		X	X		X	X	X	X	X		X	X	X	X		X		
402	219	231	196	161	78	215	47	148	93	34	131	116	24	158	223	99	X	X	X		X		X	X	X			X	X		X		X				X	X	X	X		X			X	X	X		X			X	X	X			
403	140	191	235	216	136	65	221	142	245	178	187	86	40	33	162	58		X	X	X			X	X	X			X	X	X	X	X			X	X		X	X	X	X	X			X		X	X		X		X			X	X	X
404	42	128	121	234	36	227	43	40	87	243	23	1	35	238	148	38	X	X			X	X	X	X	X						X		X						X		X	X	X		X	X	X	X	X	X	X						X
405	169	121	199	227	161	87	135	1	92	148	167	32	58	218	116	53	X	X			X		X		X			X	X	X	X	X	X		X			X	X	X			X	X	X	X	X				X	X	X	X	X		
406	63	247	37	94	170	71	98	97	91	60	220	229	115	180	237	133		X		X	X	X		X		X			X		X		X	X		X	X		X	X	X	X		X		X		X		X		X	X	X		X	X
407	44	110	221	69	94	33	84	145	7	97	152	201	25	192	229	73	X	X	X			X	X				X		X	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	X		X	X		X
408	69	145	201	73	133	122	93	213	2	108	134	23	141	107	222	63		X	X	X	X		X		X		X	X		X	X	X	X			X					X	X	X	X	X		X	X	X	X	X		X		X	X	
409	106	176	205	78	224	22	28	34	113	152	217	223	250	118	248	209		X		X	X	X			X	X			X			X			X				X	X	X			X	X	X		X		X	X	X	X	X		X	X
410	36	243	148	163	185	99	48	79	155	193	5	210	217	174	147	177	X		X		X	X		X	X	X			X	X					X		X	X	X		X	X	X			X		X		X	X	X	X	X			X
411	243	163	99	79	193	210	174	177	66	96	70	106	199	54	119	245	X				X		X	X		X	X		X	X	X	X		X	X	X	X	X				X		X		X	X	X	X				X	X		X	X
412	167	88	191	166	45	216	110	91	65	26	49	142	158	69	178	35		X		X		X	X	X			X	X		X	X	X	X	X	X	X	X	X		X	X	X				X			X		X	X	X	X	X	X	X
413	62	126	219	160	245	231	57	36	196	173	186	161	26	211	78	243		X	X	X	X		X	X	X		X	X	X		X	X		X	X	X		X		X	X		X	X		X				X	X		X				
414	151	81	118	72	83	230	172	140	163	100	135	188	42	156	137	191	X		X	X		X						X				X		X			X	X	X		X	X				X	X	X	X	X	X	X	X		X		
415	232	248	139	190	9	8	185	14	241	242	93	43	157	193	211	77	X	X	X			X	X	X	X			X	X		X				X				X	X	X		X	X		X	X	X	X	X	X	X	X		X		
416	5	70	26	237	53	56	165	17	24	30	102	76	140	233	120	169		X	X		X			X	X	X		X	X	X			X		X			X		X		X		X	X	X	X			X	X	X	X	X	X		X
417	127	51	62	61	235	126	141	101	219	65	97	160	5	9	245	28	X	X	X	X	X	X	X		X		X			X	X				X			X	X			X	X	X		X		X	X		X	X	X			X	X
418	161	148	116	99	209	155	127	210	146	147	45	66	90	51	18	106		X	X		X		X	X	X	X		X	X				X	X	X	X				X		X	X		X	X	X		X		X			X			
419	154	92	6	55	107	239	70	250	64	146	235	89	185	237	226	164		X		X	X	X	X	X	X	X		X	X		X	X		X					X	X	X		X		X	X	X	X	X			X	X		X		X
420	235	65	245	86	162	173	7	204	78	184	42	21	53	244	103	138					X	X	X		X	X	X			X		X				X		X			X			X		X	X					X	X				
421	14	77	238	25	115	3	175	143	50	46	112	49	11	179	174	141	X		X	X	X	X		X	X	X	X		X	X	X	X		X	X	X	X	X	X		X	X		X	X	X	X		X	X	X	X			X		
422	21	185	209	193	114	147	140	96	18	225	1	119	196	191	31	175	X	X	X		X		X		X	X	X		X	X	X	X		X			X		X		X	X	X		X	X	X		X	X			X	X			
423	12	5	45	70	1	26	197	237	178	53	150	56	72	165	249	17	X		X		X	X			X		X	X	X	X	X	X		X		X			X		X	X	X	X	X		X	X	X		X		X			X	
424	126	160	231	36	173	161	211	243	215	21	222	148	56	171	34	163	X			X		X	X		X		X	X		X	X		X	X		X		X				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
425	136	178	162	24	250	184	229	10	103	164	101	208	79	186	206	90	X		X	X		X	X		X	X		X	X							X	X		X		X	X	X	X	X		X		X	X	X		X			X	X
426	95	157	69	20	165	145	196	153	201	233	212	73	39	215	133	11	X			X	X			X	X	X						X							X	X	X		X	X	X	X		X	X	X	X			X		X	X
427	177	245	176	173	179	78	151	21	22	103	169	34	249	81	152	185	X	X	X	X	X	X	X				X		X		X		X	X	X			X		X		X	X	X	X	X	X	X		X		X			X	X	X
428	57	211	47	171	81	158	31	48	40	72	198	127	133	182	230	174	X		X	X		X			X	X		X		X		X		X	X		X	X		X						X				X	X	X	X	X	X	X	
429	210	106	74	176	159	205	101	78	180	224	76	22	55	28	232	34	X		X		X	X		X	X	X	X		X	X			X	X	X	X	X	X			X	X	X	X		X		X		X		X		X	X	X	
430	124	139	4	8	57	241	209	43	203	211	236	135	69	147	47	238	X	X	X	X		X		X	X	X		X		X		X		X	X		X	X	X	X		X	X			X	X		X	X		X			X	X	X
431	54	246	101	151	142	28	109	81	36	58	4	118	60	105	204	72	X				X		X		X		X	X	X			X	X	X			X	X		X	X		X		X		X		X		X		X				
432	182	39	143	220	207	141	184	80	29	221	205	9	75	208	102	109		X		X	X			X		X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X		X		X	X	X	X		X
433	96	175	159	179	64	224	142	103	232	67	219	152	155	58	220	212	X	X	X		X				X		X			X		X	X	X	X	X	X	X	X	X	X	X			X	X		X	X		X	X		X	X		X
434	15	45	136	26	32	178	94	56	162	249	54	24	230	97	250	76	X	X	X	X		X			X	X	X	X				X			X	X		X	X	X	X		X			X	X			X		X	X	X	X		X
435	152	52	190	134	109	14	137	31	43	105	116	77	180	104	68	182	X	X		X	X		X	X		X				X			X	X	X	X	X	X	X	X		X		X	X	X	X	X			X	X	X	X		X	X
436	248	190	8	14	242	43	193	77	135	68	107	238	20	96	171	25		X		X	X			X		X	X				X	X	X	X	X	X			X	X	X			X	X		X		X	X	X			X	X	X	X
437	173	21	34	185	212	209	72	193	59	114	227	147	125	140	134	96		X						X	X	X	X			X				X	X	X	X	X	X	X	X	X	X		X		X	X				X	X	X		X	X
438	51	61	126	101	65	160	9	28	231	86	133	36	70	242	173	118	X	X	X		X	X	X	X	X	X				X			X	X	X	X	X		X		X	X	X	X	X			X	X	X	X	X	X	X	X	X	X
439	28	118	243	230	138	163	115	188	99	137	85	79	169	46	193	235	X	X	X		X		X		X		X	X	X	X	X			X	X	X	X	X	X	X		X	X						X	X		X	X			X	
440	236	195	123	226	198	44	249	95	207	16	175	110	134	120	247	157	X		X		X	X	X	X	X	X		X	X	X	X	X		X	X		X	X	X	X		X			X	X			X			X	X	X			
441	83	100	137	183	154	104	189	136	96	92	127	41	36	37	6	178		X	X	X	X	X	X	X		X					X	X		X	X	X	X			X		X		X	X	X	X	X	X		X				X	X	X
442	158	127	38	51	188	62	143	61	30	235	94	126	12	141	177	101	X	X				X	X	X	X	X	X		X	X	X		X		X		X	X			X	X	X			X		X		X		X		X	X	X	X
443	32	249	250	120	66	164	202	125	206	74	142	84	104	42	98	196			X		X		X	X	X			X	X	X	X	X	X		X	X		X		X			X		X	X		X		X	X	X	X	X	X		X
444	181	27	129	232	143	124	173	248	233	141	84	139	226	21	29	190		X					X	X		X	X			X	X		X	X	X		X	X		X	X	X			X		X	X	X	X	X	X			X		X
445	251	144	128	47	28	234	14	158	227	118	213	40	229	77	243	127	X	X			X	X	X	X	X	X		X		X	X	X	X	X		X	X	X	X	X		X	X		X	X			X	X	X	X			X	X	X
446	58	83	138	100	90	137	23	183	193	154	158	104	160	189	228	136		X		X	X	X		X		X	X	X	X	X	X		X	X	X	X		X				X		X	X	X		X		X	X	X	X	X	X		X
447	142	58	204	83	10	138	122	100	185	90	47	137	126	23	149	183	X	X		X	X	X		X		X		X		X	X	X			X	X		X	X		X		X		X	X	X			X	X	X	X	X		X	X
448	131	15	183	45	87	136	247	26	41	32	174	178	118	94	55	56			X	X				X	X	X	X		X	X		X	X		X	X	X	X		X	X	X			X	X	X	X	X	X					X		
449	153	11	213	198	241	63	155	16	189	135	117	247	109	66	85	197	X		X	X				X	X	X	X		X	X	X	X			X	X	X	X			X	X		X			X	X	X		X	X			X	X	
450	73	213	23	63	203	189	146	247	88	85	182	37	242	112	5	94	X	X		X	X			X		X		X			X	X		X		X	X		X	X	X	X		X	X	X	X	X		X	X		X			X	

For each value of counter: **DataID value to be used**

For for a message with length [bytes]: **" ": not yet assigned , "X": not allowed**

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
451	30	219	125	231	176	196	144	161	192	78	200	215	178	47	22	148	X	X		X	X	X		X	X		X	X	X		X		X	X	X		X				X	X	X	X	X	X		X	X	X	X	X	X	X			
452	7	244	81	2	200	72	82	167	230	131	43	140	251	123	100	88	X	X	X	X	X		X				X	X		X		X	X	X	X	X	X	X	X		X	X				X	X	X	X	X	X	X	X	X			X
453	178	24	184	10	164	208	186	90	212	84	28	149	177	222	214	154		X	X	X	X			X		X	X		X		X	X	X	X	X	X		X	X				X	X		X					X	X		X		X	X
454	132	129	165	124	49	233	78	139	133	29	214	4	44	34	251	8		X	X	X	X	X	X		X	X	X			X	X	X	X	X	X	X	X	X					X		X	X		X	X	X	X			X	X	X	
455	237	17	76	169	219	217	4	121	90	231	244	199	216	241	196	227			X	X		X		X	X	X		X		X		X			X		X	X	X		X	X	X	X	X	X	X					X			X	X	
456	214	19	168	236	153	82	87	195	172	11	193	123	152	32	213	226	X					X	X							X	X		X	X			X	X	X			X	X	X	X	X	X			X	X				X		X
457	46	150	54	102	166	246	130	7	101	91	124	151	218	240	142	244	X				X		X	X	X	X	X		X		X	X	X		X	X		X			X		X	X	X			X		X	X	X		X			X
458	75	132	197	129	3	165	176	124	97	49	206	233	123	78	187	139	X		X	X	X	X	X	X	X			X		X								X		X	X	X		X	X	X	X	X		X			X	X	X	X	
459	183	136	41	178	55	162	71	24	179	250	61	184	163	229	89	10	X			X		X	X		X					X			X	X			X	X	X	X	X	X	X		X	X	X	X	X	X	X			X		X	X
460	68	115	48	46	167	174	64	150	51	88	132	54	241	67	191	102			X		X	X				X	X	X	X	X		X		X	X	X		X	X		X				X	X		X	X			X			X	X	
461	67	130	80	240	69	109	90	168	242	145	34	105	27	154	201	82			X			X		X		X	X				X	X	X	X	X	X			X	X	X	X		X	X		X		X		X					X	X
462	199	87	92	32	116	55	170	249	239	155	191	250	138	60	146	120		X	X		X	X				X	X	X	X			X	X	X	X	X	X		X		X	X			X	X	X	X		X		X	X	X		X	X
463	2	167	140	88	15	191	44	166	235	45	3	216	47	110	136	91	X						X		X	X		X	X	X	X					X		X	X		X			X	X	X	X	X	X			X	X		X	X	
464	121	227	87	1	148	32	218	53	55	99	88	249	83	50	155	30			X		X	X	X	X		X	X	X	X	X	X			X	X		X	X		X	X	X	X	X	X			X		X			X		X	X	
465	221	33	7	201	229	244	19	122	81	186	190	2	49	236	200	23	X	X	X		X	X	X			X		X	X			X		X	X		X			X	X	X	X					X	X			X	X	X	X		
466	191	216	65	142	178	86	33	58	173	24	251	204	38	201	184	83	X			X	X		X		X	X	X	X	X	X	X	X	X	X		X		X	X	X	X	X	X	X					X				X	X	X	X	X
467	81	72	230	140	100	188	156	191	79	183	218	235	128	207	104	216		X	X		X			X	X		X	X	X		X	X	X			X		X	X	X		X	X	X		X			X		X	X	X				X
468	88	166	216	91	26	142	69	35	86	56	29	58	127	145	24	200		X	X		X	X	X	X	X	X	X	X	X				X	X			X	X	X	X		X		X		X	X		X			X			X	X	X
469	60	202	17	42	126	169	139	128	217	160	201	121	166	8	231	234	X	X				X	X	X	X	X	X		X	X			X		X			X	X		X			X				X	X	X		X			X		
470	201	122	2	23	222	167	195	189	140	12	77	88	57	226	15	37	X	X	X	X	X		X		X			X				X		X	X	X	X		X	X	X	X	X	X	X	X		X	X		X		X	X	X	X	
471	157	20	145	153	233	73	215	11	122	4	114	213	220	116	108	198		X					X		X	X	X			X	X	X	X	X	X	X	X	X	X			X			X		X	X		X	X			X	X	X	
472	115	46	174	150	88	54	67	102	61	166	129	246	135	130	216	7						X	X		X				X	X	X		X		X		X		X			X	X		X	X				X	X	X		X	X	X	X
473	48	174	51	54	191	61	220	246	126	216	165	101	85	80	65	151	X	X		X	X		X		X				X	X	X	X		X		X			X	X	X	X	X		X	X	X	X					X		X	X	X
474	206	214	240	19	20	168	199	236	105	153	185	82	224	87	73	195		X	X		X	X		X		X	X	X		X		X		X	X	X	X	X	X								X	X	X			X		X	X		X
475	145	73	122	213	108	23	107	63	167	203	31	189	9	146	12	247	X	X	X	X	X	X	X	X	X	X	X			X	X		X	X		X	X		X	X	X	X	X	X			X	X	X	X		X	X				X
476	245	173	78	21	103	34	81	185	223	212	121	209	120	72	52	193	X	X			X	X		X	X	X	X		X		X	X	X	X	X	X		X			X		X				X	X		X		X		X	X	X	X
477	128	234	227	40	243	1	238	38	32	163	189	53	200	3	99	62			X	X				X		X	X		X	X		X	X		X			X	X	X	X		X	X			X		X	X				X	X		X
478	170	60	237	202	62	17	124	42	76	126	33	169	88	139	219	128	X	X	X		X	X			X					X				X	X	X			X		X		X			X			X	X	X	X	X	X	X	X	X
479	78	34	223	209	52	59	230	147	194	134	87	18	84	188	14	119	X		X	X		X	X		X	X	X		X				X			X	X		X	X		X	X	X		X			X		X	X	X			X	X
480	230	188	79	235	104	177	150	65	106	41	60	245	227	102	175	86		X	X		X	X			X		X	X				X	X			X	X		X			X				X		X	X	X	X	X	X	X	X		X
481	110	69	33	145	97	201	192	73	244	133	52	122	143	93	186	213		X			X	X	X	X	X	X		X	X		X	X	X	X			X	X	X	X		X			X	X	X			X		X	X		X		
482	155	66	112	74	181	98	126	205	157	27	24	180	6	160	129	22	X	X	X	X		X		X	X				X				X			X				X	X	X		X	X		X	X	X	X		X	X		X	X	X
483	215	116	107	155	59	146	38	66	226	18	136	112	149	62	75	74	X	X			X		X	X	X	X			X			X	X	X	X				X	X	X	X	X	X				X	X	X	X	X	X	X			X
484	84	192	19	93	113	236	227	107	82	13	137	195	103	1	11	146	X		X	X	X	X		X		X	X	X	X	X		X	X	X	X	X		X	X	X	X		X	X		X	X	X		X			X		X	X	X
485	91	35	58	200	76	83	73	131	138	217	211	100	61	213	90	15	X	X		X		X	X	X			X		X	X	X	X			X	X		X		X			X					X	X	X	X	X			X	X	X
486	175	179	224	103	67	152	58	212	248	130	231	52	66	83	80	114	X	X	X	X			X		X	X	X	X	X	X	X		X		X			X		X					X	X		X	X	X	X	X	X	X	X	X	X
487	148	99	155	210	147	66	51	106	112	119	26	74	154	61	181	176	X	X		X	X			X	X			X	X					X			X	X	X		X	X	X	X	X	X				X	X	X				X	X
488	204	138	185	137	149	193	167	104	147	228	40	96	231	88	225	41	X		X	X	X	X	X		X			X	X	X		X		X	X	X	X	X	X	X		X	X		X	X	X	X		X	X	X		X	X	X	X
489	113	13	11	194	8	198	99	75	63	43	6	16	240	210	135	132	X			X	X	X	X	X	X				X		X	X	X		X			X		X		X	X	X	X				X			X	X	X		X	X
490	10	90	149	154	192	228	12	92	225	93	230	6	173	5	236	55	X		X				X	X	X		X	X	X	X	X	X	X		X	X	X	X	X			X	X	X	X			X	X	X				X	X	X	
491	112	98	157	180	129	20	231	113	145	124	208	153	64	161	233	13	X	X		X	X	X	X	X	X	X	X		X	X	X			X		X	X	X			X	X	X		X			X		X	X	X	X			X	
492	74	205	180	22	232	113	36	223	153	248	90	13	89	243	139	59	X	X	X	X			X	X					X	X	X	X	X		X			X	X	X	X			X	X		X				X	X		X		X	X
493	26	56	24	76	120	10	133	217	208	125	151	90	235	108	84	199		X			X	X	X				X	X		X	X	X	X	X		X		X	X	X			X			X			X					X	X	X	X
494	220	80	9	109	33	242	149	105	211	201	223	68	129	228	244	172		X			X	X	X	X			X		X	X		X	X	X		X		X	X	X			X	X			X		X	X	X	X			X	X	
495	196	215	93	116	223	107	40	155	195	59	183	146	208	38	194	66	X	X	X		X	X			X	X			X	X	X		X			X	X	X		X	X	X		X		X	X	X	X	X		X	X				
496	85	170	70	60	38	237	129	202	56	62	221	17	167	124	30	42				X	X	X	X	X			X			X		X		X	X	X		X		X	X		X	X	X								X	X	X	X	
497	130	240	109	168	145	105	154	82	68	73	209	172	232	92	122	123	X			X	X	X	X		X	X	X					X	X	X	X	X			X	X			X	X			X	X	X	X	X	X		X	X		X
498	52	134	14	31	105	77	104	182	238	172	155	25	113	41	115	39	X		X	X	X	X			X		X		X	X				X		X		X	X	X	X		X	X	X	X		X	X		X	X	X			X	X
499	59	18	75	181	77	132	177	27	197	25	250	129	236	245	3	232	X	X	X		X	X		X	X	X		X	X	X	X	X	X	X			X	X	X		X	X	X		X		X	X	X	X	X	X	X				X
500	143	141	29	9	102	57	212	242	144	7	113	211	197	114	151	68	X	X		X		X			X		X				X	X			X	X	X		X	X	X		X	X	X				X	X	X	X		X	X	X	X

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)