Document Title	Specification of SW-C End-to-End Communication Protection Library
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	428
Document Classification	Standard

Document Version	2.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
21.12.2011	2.0.0	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
19.10.2010	1.1.0	AUTOSAR Administration	Corrected the wrapper configuration. Corrected the code example for the usage of the wrapper.

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
17.12.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

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).

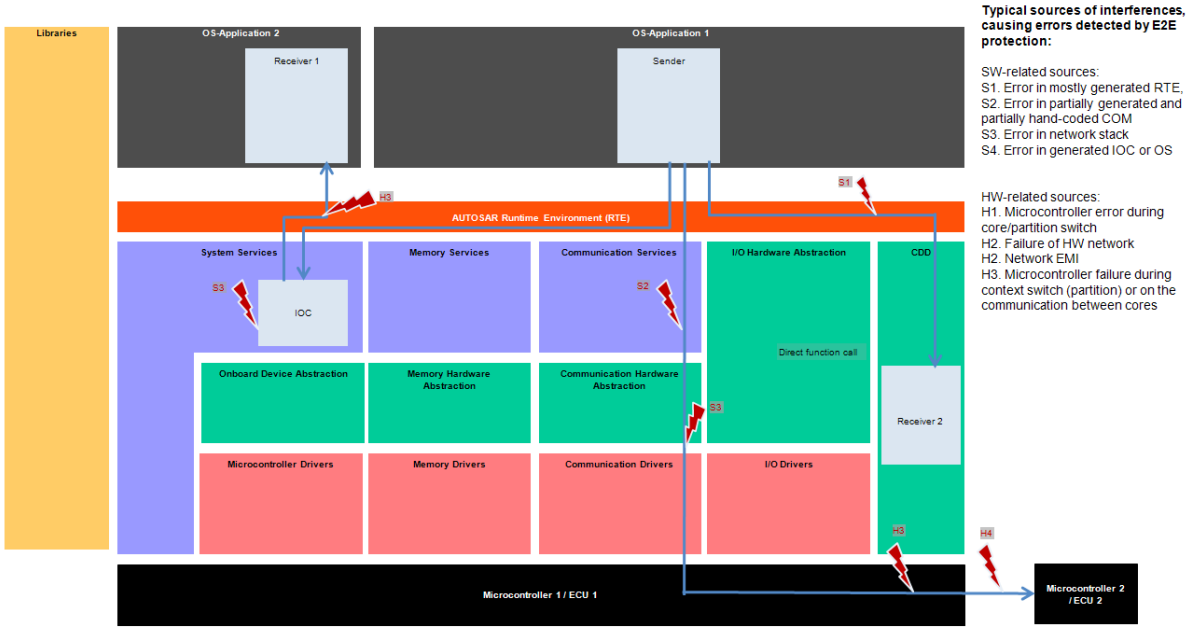


Figure 1-1: Example of faults mitigated by E2E protection

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).

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

2 Acronyms and abbreviations

All technical terms used in this document, except the ones listed in the table below, can be found in the official AUTOSAR glossary [10].

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

Abbreviation / Acronym:	***Description:***
E2E Library	Short name for SW-C End-to-End Communication Protection Library
Data ID	An identifier that uniquely identifies the message / data element / data.
Safety protocol fault model	A model of the faults for the safety communication protocol that is able to foresee the consequences of each fault.
Repetition	Repeatedly sending of the same data.
Deletion	Loss of data or parts of data during transmission.
Insertion	Unintentionally inserted data having correct source- and destination address.
Incorrect sequence	The data is not received in the same order as in which it has been sent.
Corruption	One or more bits of a data element are changed during transmission.
Timing faults (delay)	The timing constraints for the communication between sender and receiver are violated (e.g., data is received too late).
Addressing faults	(1) A non-authentic message is designed thus to appear to be authentic (an authentic message means a valid message in which the information is certificated as originated from an authenticated data source) (2) A message is accepted from an incorrect sender or by an incorrect receiver.
Inconsistency	When communicating peers have different view of network status or different data to be transmitted.

Table 2-1: Acronyms and abbreviations

In the whole document, there are many requirements that apply to all E2E Profiles at the same time (i.e. to E2E Profile 01 and E2E Profile 02). Such requirements are defined as one requirement that applies to all profiles at the same time. In case some names are profile dependent, then XX notation is used: if in a requirement appears the string containing XX, then it is developed to two strings with 01 and 02 respectively instead of XX. For example, E2E_PXXCheck() develops to the following two E2E_P01Check(), E2E_P02Check().

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] AUTOSAR Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of COM
AUTOSAR_SWS_COM.pdf

[5] Specification of BSW Scheduler
AUTOSAR_SWS_Scheduler.pdf

[6] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

[7] Specification of CRC Routines
AUTOSAR_SWS_CRCLibrary.pdf

[8] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[9] Requirements on Libraries
AUTOSAR_SRS_Libraries.pdf

[10] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf

[11] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[12] System Template
AUTOSAR_TPS_SystemTemplate.pdf

[13] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

3.2 Related standards and norms

[14] ISO DIS 26262 (reference safety standard for AUTOSAR)
http://www.iso.org/

[15] EN 50159
http://www.cenelec.eu/

4 Constraints and assumptions

4.1 Limitations

E2E Profile 1 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 E2EUSE072 and E2EUSE073.

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

[E2E0224] ⌈
If the E2E Library is invoked at the level of Data Elements (e.g. from SW-Cs or from E2E Protection Wrapper), then the communication shall be an explicit sender-receiver communication, in 1:1 and 1:N multiplicities.

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

[E2E0255] ⌈
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. ⌋ ( )

[E2E0226] ⌈
If the E2E Library is invoked at the level of Data Elements, then a Data Element shall either map to a local intra-ECU communication (without COM involvement) or shall map to a COM I-PDU, but shall not map to both at the same time. ⌋ ( )

4.1.2 Limitations when invoking library at the level of I-PDUs

[E2E0269] ⌈
If the E2E library is invoked at the level of I-PDUs, then there shall be only one set of settings for an I-PDU.

The above means that it is not possible to have several signal groups protected by E2E in an I-PDU by means of callouts. ⌋ ( )

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 needed for inter-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

In order to ensure freedom from interference by software partitioning, measures to detect the following communication faults are taken into account (see ISO 26262 [14]):

- loss of peer to peer communication;
- unintended message repetition due to the same message being unintentionally sent again;
- message loss during transmission;
- insertion of messages due to receiver unintentionally receiving an additional message, which is interpreted to have correct source and destination addresses;
- re-sequencing due to the order of the data being changed during transmission, i.e. the data is not received in the same order as in which it was been sent;
- message corruption due to one or more data bits in the message being changed during transmission;
- message delay due to the message being received correctly, but not in time;
- blocking access to data bus due to a faulty node not adhering to the expected patterns of use and making excessive demands for service, thereby reducing its availability to other nodes, e.g. while wrongly waiting for non existing data; and

- constant transmission of messages by a faulty node, thereby compromising the operation of the entire bus.

The E2E Library implements such measures.

4.3.2 Sources of faults in E2E communication

E2E protection aims to mitigate the effects of communication faults arising from:

1. (systematic) software faults,
2. (random) hardware faults,
3. as well as 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 Summary of fault model

The following faults related to message exchange via communication network have been considered (as reference see EN 50159 [15] and ISO 26262 [14]).

4.3.3.1 Repetition

This means that the same message is received more than once. Repetition cannot be neglected in any communication system (see e.g. EN 50159-2 table C.2).

4.3.3.2 Deletion

This means that the message or parts of it have been removed from the communication stream. Deletion cannot be neglected in any communication system (see e.g. EN 50159-2 table C.2).

4.3.3.3 Insertion

This means that an additional message or parts of it have been inserted into the communication stream. For vehicle internal networks (e.g. CAN, FlexRay) the occurrence rate of this fault is not high and weak counteractive measures are sufficient (see e.g. EN 50159-2 table C.2).

4.3.3.4 Incorrect sequence

This means that messages of a communication stream are received in an incorrect order. For vehicle internal networks (e.g. CAN, FlexRay) the occurrence rate is not high and weak counteractive measures are sufficient (see e.g. EN 50159-2 table C.2).

4.3.3.5 Corruption

This means that the corruption data of a message or parts of it occurred. Corruption is one of the most common error types and it cannot be neglected in any communication system[1] (see e.g. EN 50159-2 table C.2)

[1] Note that if the bit corruption occurs in the payload of a safety message (i.e. not in the header containing e.g. checksum or address), then this is not likely to cause any other protocol failures. In contrary, if the corruption occurs in the control fields, then it is detected also by other protection measures. For example, if the corruption occurs in the sequence number, you will experience this is repetition, insertion, or incorrect sequence.

4.3.3.6 Timing faults (delay)

This means that the timing constraints of a message are violated (e.g. the message is received too late) . It cannot be neglected in any communication system.

4.3.3.7 Addressing faults

Due to faults, a message is sent to the wrong destination, which then treats reception as correct.

4.3.3.8 Inconsistency

Due to faults, communicating nodes have a different view of network status or of data being transferred.

4.3.3.9 Masquerading

This means that the design of a received message with non-authentic content appears authentic as just sent by the appropriate sender.

4.4 Implementation of the E2E Library

[E2E0050] ⌈
The implementation of the E2E Library shall comply with the requirements for the development of safety-related software for the automotive domain.

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. ⌋ (BSW08527)

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.

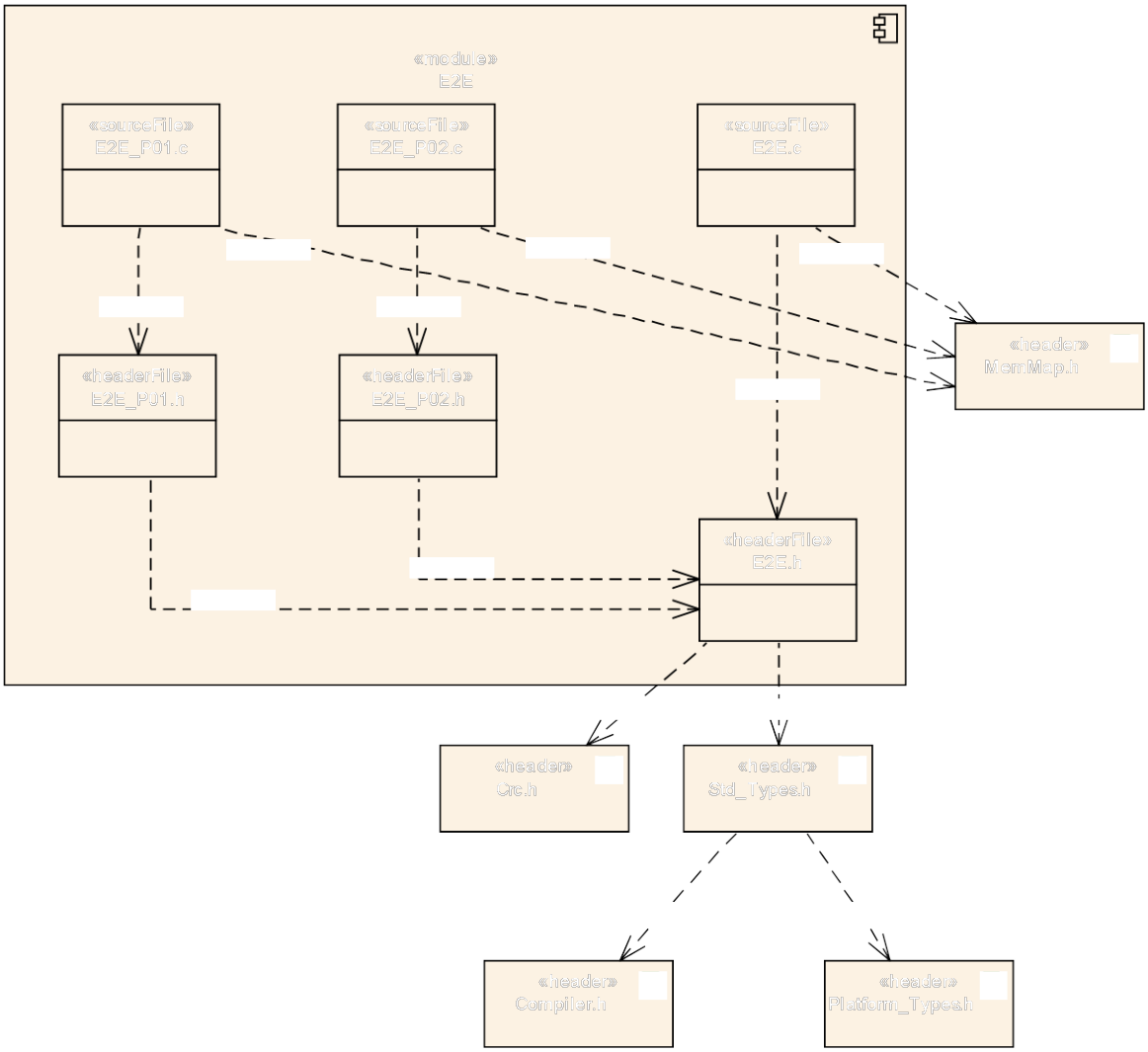


Figure 5-1: File dependencies

[E2E0048] ⌈
E2E library shall be built of the following files: E2E.h (common header), E2E.c (implementation of common parts), E2E_PXX.c and E2E_PXX.h (where XX: 01, 02).
⌋ ( )

[E2E0215] ⌈
Files E2E_PXX.c and E2E_PXX.h shall contain implementation parts specific of each profile. ⌋ ( )

[E2E0111] ⌈
E2E.h shall exclusively include: CRC.h, Std_Types.h, PlatformTypes.h, MemMap.h. E2E.h shall not include any other files. ⌋ (BSW00436)

[E2E0112] ⌈
E2E.c shall exclusively include E2E.h. ⌋ ( )

[E2E0113] ⌈
E2E_PXX.h shall exclusively include E2E.h. ⌋ ( )

[E2E0114] ⌈
Each E2E_PXX.c file shall exclusively include the corresponding E2E_PXX.h file. ⌋ ( )

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

[E2E0115] ⌈
E2E library files (i.e. E2E_*.*) shall not #include any RTE files. ⌋ ( )

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

5.1.2 Dependency on CRC library

It is important to note that the function Crc_CalculateCRC8 of CRC library / CRC routines have changed 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 R4.1 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 7-5) in >=R4.0 and R3.2.

6 Requirements traceability

Requirement	Satisfied by
-	E2EUSE0240
-	E2E0148
-	E2EUSE0296
-	E2EUSE0250
-	E2E0128
-	E2EUSE0273
-	E2EUSE205
-	E2E0169
-	E2E0154
-	E2EUSE087
-	E2E0095
-	E2EUSE204
-	E2EUSE0300
-	E2E0160
-	E2E0136
-	E2E0158
-	E2EUSE0259
-	E2E0107
-	E2EUSE0290
-	E2EUSE057
-	E2EUSE0280
-	E2E0132
-	E2E0138
-	E2E0150
-	E2E0085
-	E2E0119
-	E2EUSE0261
-	E2EUSE0213
-	E2E0145
-	E2E0121
-	E2E0092
-	E2EUSE051
-	E2E0200
-	E2EUSE061

-	E2EUSE0270
-	E2E0123
-	E2E0134
-	E2E0122
-	E2EUSE0241
-	E2EUSE0268
-	E2E0215
-	E2E0224
-	E2E0033
-	E2E0120
-	E2E0091
-	E2E0112
-	E2E0143
-	E2EUSE053
-	E2E0113
-	E2E0196
-	E2E0098
-	E2EUSE0277
-	E2E0226
-	E2EUSE170
-	E2E0118
-	E2E0130
-	E2E0140
-	E2E0133
-	E2E0195
-	E2EUSE089
-	E2EUSE056
-	E2EUSE236
-	E2E0190
-	E2E0020
-	E2E0146
-	E2E0129
-	E2E0149
-	E2EUSE173
-	E2E0048
-	E2EUSE062
-	E2EUSE0192
-	E2EUSE0251

-	E2EUSE203
-	E2E0124
-	E2EUSE0256
-	E2E0127
-	E2E0161
-	E2E0076
-	E2EUSE0263
-	E2E0097
-	E2EUSE0257
-	E2EUSE209
-	E2E0126
-	E2EUSE0272
-	E2EUSE0239
-	E2EUSE0301
-	E2EUSE0258
-	E2EUSE055
-	E2E0142
-	E2E0135
-	E2E0094
-	E2E0163
-	E2E0166
-	E2E0151
-	E2EUSE072
-	E2EUSE207
-	E2E0227
-	E2E0083
-	E2E0115
-	E2E0114
-	E2E0017
-	E2E0021
-	E2EUSE0260
-	E2E0110
-	E2E0012
-	E2E0096
-	E2E0153
-	E2E0125
-	E2E0141
-	E2E0106

-	E2EUSE233
-	E2EUSE0274
-	E2EUSE202
-	E2E0099
-	E2E0137
-	E2EUSE071
-	E2EUSE237
-	E2EUSE0165
-	E2EUSE0264
-	E2EUSE0267
-	E2EUSE206
-	E2EUSE0297
-	E2EUSE235
-	E2EUSE230
-	E2EUSE063
-	E2EUSE0249
-	E2EUSE0292
-	E2EUSE0271
-	E2E0276
-	E2E0269
-	E2E0288
-	E2E0139
-	E2E0152
-	E2E0255
-	E2EUSE073
-	E2EUSE0275
-	E2EUSE0266
-	E2EUSE0265
-	E2E0018
-	E2EUSE208
-	E2EUSE0262
-	E2E0293
-	E2EUSE0242
-	E2EUSE0279
-	E2E0147
-	E2E0075
-	E2EUSE0248
-	E2EUSE0289

-	E2E0228
-	E2E0287
-	E2E0011
BSW003	E2E0032
BSW00323	E2E0047
BSW00336	E2E0294
BSW00337	E2E0047
BSW00338	E2E0049, E2E0294
BSW00339	E2E0294, E2E0216
BSW00369	E2E0049, E2E0294
BSW00375	E2E0294
BSW00435	E2E0294
BSW00436	E2E0111
BSW08527	E2E0050
BSW08528	E2E0217
BSW08529	E2E0219, E2E0218
BSW08531	E2E0221, E2E0117, E2E0070
BSW08533	E2E0219, E2E0218
BSW08534	E2E0047, E2E0022, E2E0214
BSW08536	E2E0082
BSW168	E2E0294

7 Functional specification

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

7.1 Overview of communication protection

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

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

Moreover, some E2E Profiles have standard variants. A 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 variant 1A requires that CRC starts at bit 0 and counter starts at bit 8.

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

E2E protection uses the following safety mechanisms:

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

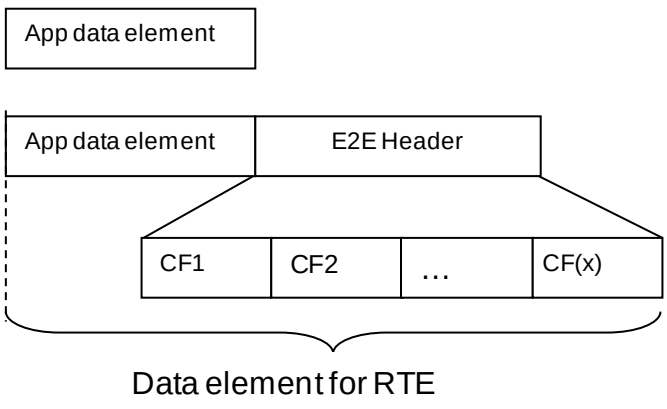


Figure 7-1: Safety protocol concept (with exemplary location of the E2E header)

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

E2E protection can be used at the level of Data Elements, or at the level of COM I-PDUs. If used at the level of data elements, then there are some restrictions, as described in Section 4.1.

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.

[E2E0221] ⌈
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 (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. ⌋ (BSW08531)

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 used: 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.

[E2E0217] ⌈
The implementation of the E2E Library shall provide at least one of the E2E Profiles, i.e. E2E Profile 1 or E2E Profile 2. ⌋(BSW08528)

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. Also, 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.

[E2E0049] ⌈
The E2E library shall not contain library-internal mechanisms for error detection to be traced as development errors. ⌋(BSW00338, BSW00369)

[E2E0011] ⌈
The E2E Library shall report errors detected by library-internal mechanisms to callers of E2E functions through return value. ⌋( )

[E2E0216] ⌈
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. ⌋(BSW00339)

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

[E2E0047] ⌈
The following error flags for errors shall be used by all E2E Library functions. The functions E2E_P01Protect(), E2E_P01Check(), E2E_P02Protect(), E2E_P02Check() shall use these values as return value:

Type or error or status	Relevance	Related code	Value [hex]
At least one pointer parameter is a NULL pointer	Production	E2E_E_INPUTERR_NULL	0x13
At least one input parameter	Production	E2E_E_INPUTERR_WRONG	0x17

is erroneous, e.g. out of range			
An internal library error has occurred (e.g. error detected by program flow monitoring, violated invariant or postcondition)	Production	E2E_E_INTERR	0x19
Function completed successfully	N/A	E2E_E_OK	0x00
Invalid value	Production	E2E_E_INVALID	0xFF

Table 7-1: Return values of E2E Library functions

⌋(BSW00337, BSW00323, BSW08534)

Note that the E2E Protection Wrapper (code examples) uses the same byte to store E2E and E2EPW errors. Therefore, some value ranges are reserved.

E2E0295:
The functions E2E_P01Protect(), E2E_P01Check(), E2E_P02Protect(), E2E_P02Check() shall never return values 0x1 and values in the range 0x80 .. 0xFF.

7.2.2 Error detection

[E2E0012] ⌈
The library-internal mechanisms shall detect and report errors shall be implemented according to the pre-defined E2E Profiles specified in sections 7.3 and 7.4. ⌋( )

7.3 Specification of E2E Profile 1

Profile 1 shall provide the following mechanisms:

[E2E0218] ⌈

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	Timeout is determined by E2E Library by means of evaluation of the Counter, by a non-blocking read at the receiver. Timeout is reported by E2E Library to the caller by means of the status flags in E2E_P01ReceiverStatusType
Data ID	16 bit, unique number, included in the CRC calculation but not transmitted (implicit transmission)

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 R4.0, the SAE8 CRC function of the CRC library uses 0xFF as start value and XOR value. To compensate a different behavior of the CRC library, the E2E Library applies additional XOR 0xFF operations starting with R4.0, to come up with 0x00 as start value and XOR value. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN.

Table 7-2: E2E mechanisms

The mechanisms are used to mitigate the following failure modes:

Mechanism	Detected failure modes
Counter	Repetition, deletion, insertion, incorrect sequence
Timeout	Deletion, delay
Data ID	Insertion, addressing faults
CRC	Corruption

Table 7-3: E2E mechanisms vs. failure modes

⌋(BSW08529, BSW08533)

[E2E0070] ⌈
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 shall be 0x00. ⌋(BSW08531)

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 1A and 1B 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.

[E2E0075] ⌈
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. ⌋ ( )

[E2E0076] ⌈
In E2E Profile 1, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following shall be detected: (1) no new data has arrived since last invocation of E2E library check function, (2) no new data has arrived since receiver start, (3) the data is repeated (4) counter is incremented by one (i.e. no data lost), (5) counter is incremented more than by one, but still within allowed limits (i.e. some data lost), (6) counter is incremented more than allowed (i.e. too many data lost). All these actions shall be executed by E2E Library.

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

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

7.3.3 Data ID

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

[E2E0085] ⌈
In E2E Profile 1, the length of the Data ID shall be 16 bits (i.e. 2 byte). ⌋ ( )

[E2E0169] ⌈
If in a given system the high byte of Data ID is unused (E2E_P01DataIDMode = E2E_P01_DATAID_LOW), then the high byte of Data ID shall be set to 0x00. ⌋ ( )

In E2E Profile 1, the Data ID is transmitted implicitly. This means that Data ID is not transmitted together with the data, but it is included in the CRC calculation.

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 will receive it only from one sender. In case of usage of E2E Library for protecting I-PDUs, due to the fact that each I-PDU has a unique Data ID, the receiver COM of an I-PDU will receive it from only from one sender COM. As a result, 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.

In E2E Profile 1, there are three ways (inclusion modes) how the Data ID is included into the CRC calculation: either both bytes are included in CRC, or alternating (depending on parity of counter) either the high or the low byte is used, or only the low byte is included into the CRC.

[E2E0163] ⌈
There shall be following three inclusion modes for the two-byte Data ID into the calculation of the one-byte CRC:

1. both two bytes (double ID configuration) are included in the CRC, first low byte and then high byte (see variant 1A - E2E0227) or
2. depending on parity of the counter (alternating ID configuration) the high and the low byte is included (see variant 1B - E2E0228). For even counter values the low byte is included and for odd counter values the high byte is included.
3. 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. ⌋ ( )

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.

[E2E0083] ⌈
E2E Profile 1 shall use CRC-8-SAE J1850 for CRC calculation. It shall use 0x00 as the start value and XOR value. ⌋ ( )

[E2E0190] ⌈
E2E Profile 1 shall use the Crc_CalculateCRC8 () function of the SWS CRC Library for calculating CRC checksums. ⌋ ( )

The Data ID itself is not transmitted in E2E Profile 1, but it is included in the CRC. This is called implicit transmission of Data ID (note: this is a different notion than implicit communication of data Elements through RTE).

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 example (for standard variant 1A, and for a particular set of signals):

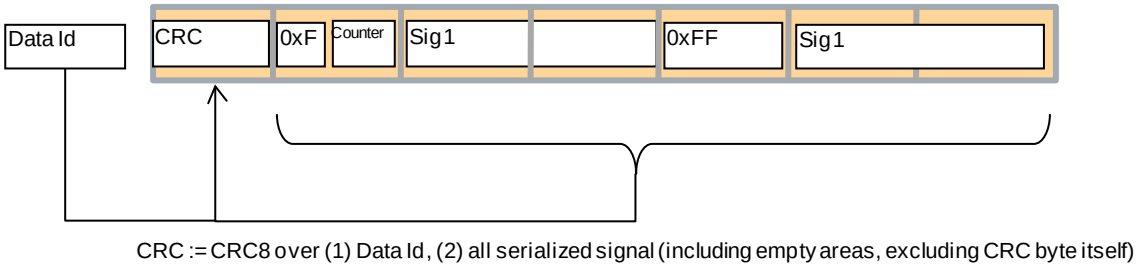


Figure 7-2: E2E Profile 1 CRC calculation example

The Data ID can be encoded in CRC in different ways, see E2E0163.

In E2E Profile 1, the CRC is calculated over:

1. First over the one or two bytes of the Data ID (depending on configuration) and then
2. Then over all transmitted bytes of a safety-related complex data element/signal group (except the CRC byte).

[E2E0082] ⌈
In E2E Profile 1, the CRC shall be calculated, with start value 0x00 over one or two bytes of the Data ID (depending on the configuration according to E2E0163). ⌋ (BSW08536)

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 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_P01STATUS_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.

7.3.6 E2E Profile 1 variants

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

[E2E0227] ⌈
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. Both bytes of Data ID are included every time the CRC of the data is computed (E2E_P01DataIDMode = E2E_P01_DATAID_BOTH). ⌋ ( )

[E2E0228] ⌈
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. Only one byte of Data ID is included at a time when the CRC of the data is computed (E2E_P01DataIDMode = E2E_P01_DATAID_ALTERNATING). ⌋ ( )

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

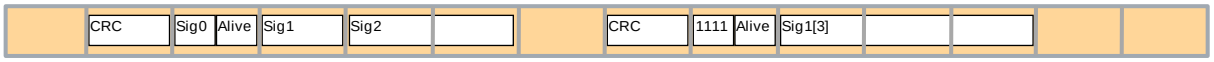


Figure 7-3: E2E Profile 1 example layout (two Signal Groups protected by E2E in one I-PDU)

7.3.7 E2E_P01Protect

[E2E0195] ⌈
The function E2E_P01Protect() shall write the Counter in Data, an finally compute the CRC over DataID and Data and write CRC in Data. Then it shall increment the Counter (which will be used in the next invocation of E2E_P01Protect()), as specified by Figure 7-4 and Figure 7-5. ⌋ ( )

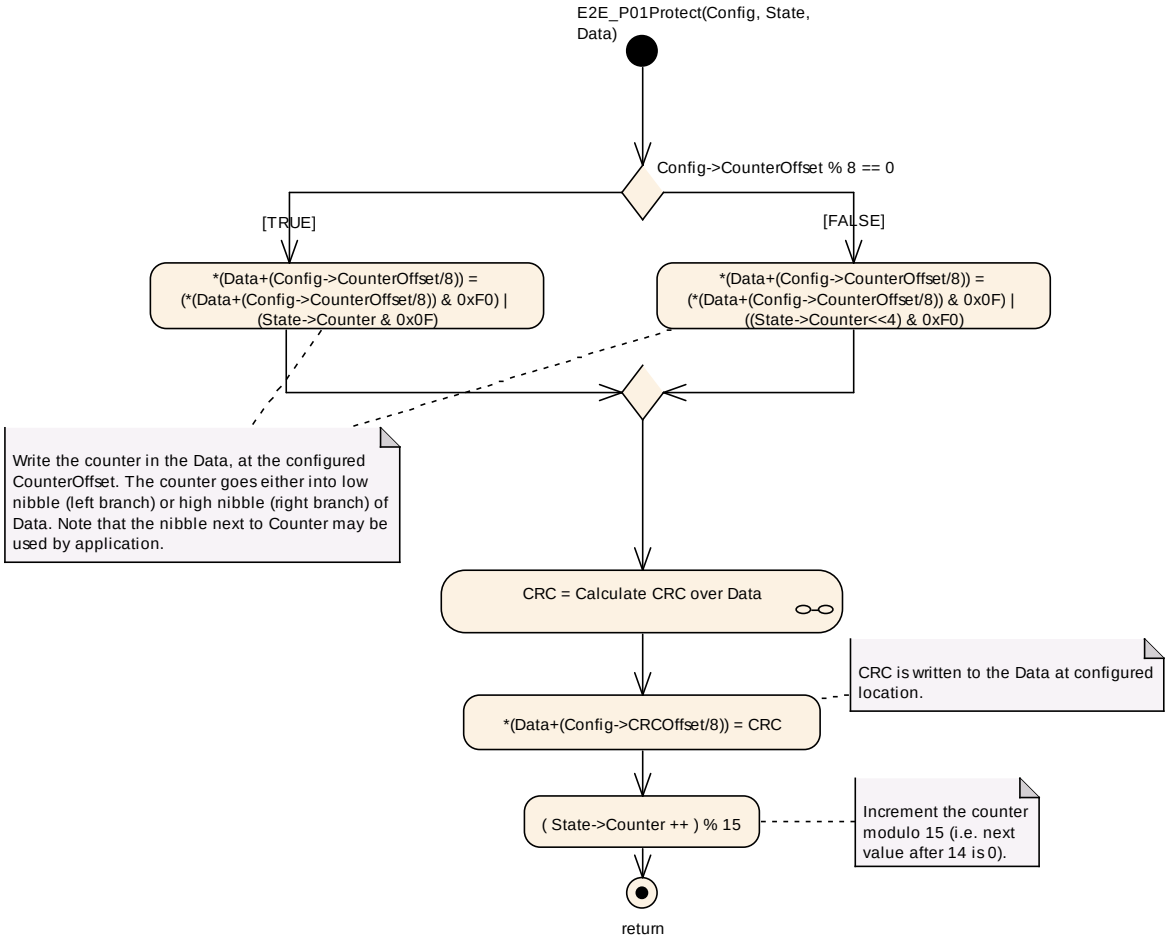


Figure 7-4: E2E_P01Protect()

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:

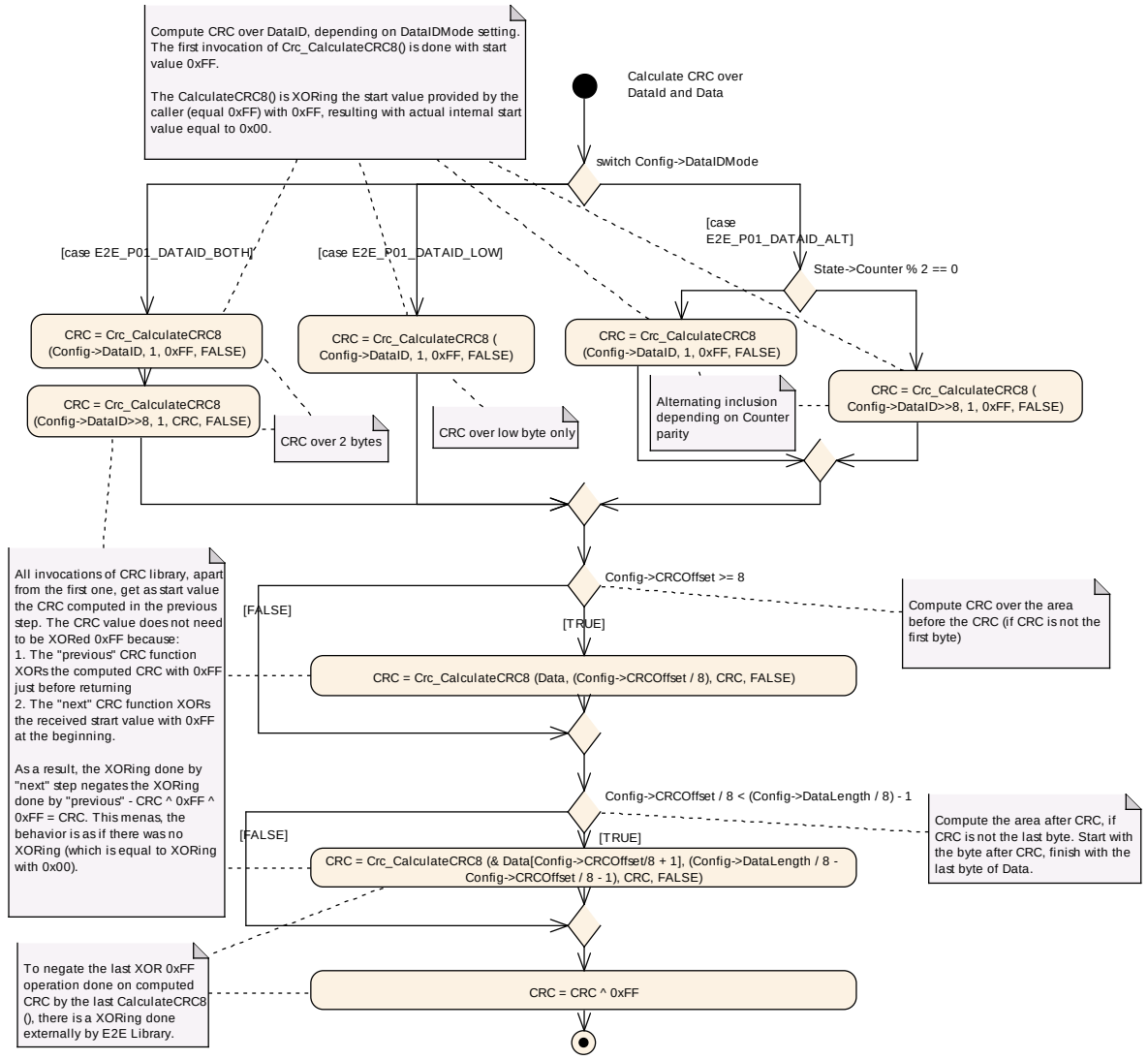


Figure 7-5: Subdiagram “Calculate CRC over Data ID and Data”, used by E2E_P01Protect() and E2E_P01Check()

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 R4.1 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 7-5) in >=R4.0 and R3.2.

7.3.9 E2E_P01Check

[E2E0196]⌈
The function E2E_P01Check shall check the Counter and CRC of the received Data and determine the check Status, as specified by Figure 7-6 and Figure 7-5.⌋( )

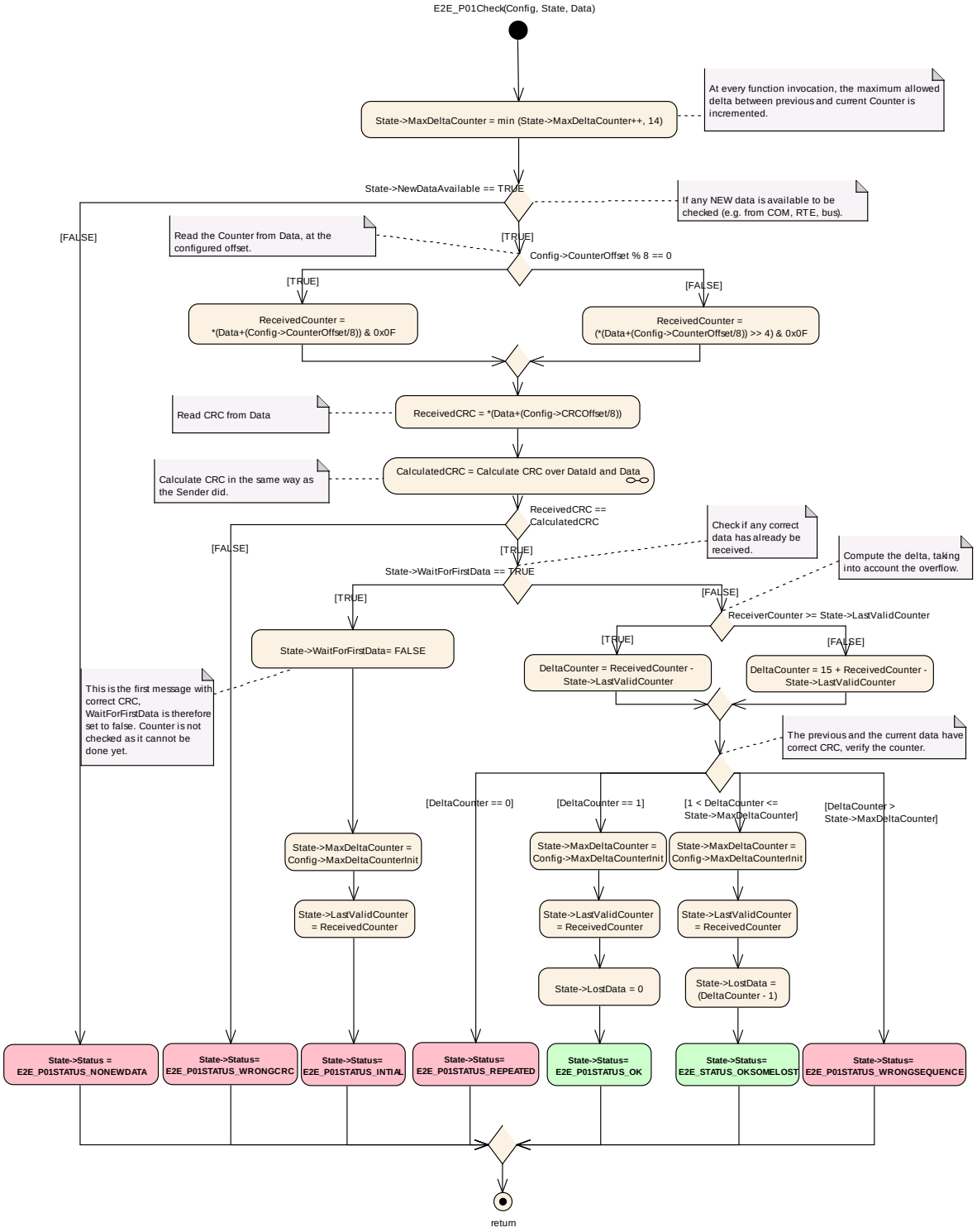


Figure 7-6: E2E_P01Check()

The diagram of the function E2E_P01Check() has a sub-diagram specifying the calculation of CRC, which is shown by Figure 7-5.

7.4 Specification of E2E Profile 2

[E2E0219] ⌈
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.

Table 7-4: E2E Profile 2 mechanisms

⌋ (BSW08529, BSW08533)
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 following table gives an overview of the detectable failure modes with respect to the mechanisms:

Mechanism	Detected failure modes
Counter	Unintended message repetition, message loss, insertion of messages, re-sequencing
Data ID	Insertion of messages, masquerading

CRC	Message corruption, insertion of messages (masquerading)
Timeout (detection and handling implemented by SW-C)	Message loss, message delay

Table 7-5: Detected failure modes using Profile 2

[E2E0117] ⌈
E2E Profile 2 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Library for calculating CRC checksums. ⌋(BSW08531)

[E2E0118] ⌈
E2E Profile 2 shall use 0xFF as the start value CRC_StartValue8 for CRC calculation. ⌋( )

[E2E0119] ⌈
In E2E Profile 2, the specific Data ID used to calculate a specific CRC shall be of length 8 bit. ⌋( )

[E2E0120] ⌈
In E2E Profile 2, the specific Data ID used for CRC calculation shall be selected from a pre-defined DataIDList[16] using the value of the Counter as an index. ⌋( )

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

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

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

An example DataIDList is presented in Chapter 13.4.

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

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

[E2E0121] ⌈

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

Figure 7-7: Data of E2E Profile 2

⌋( )

[E2E0122] ⌈

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

[E2E0123] ⌈

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

[E2E0124] ⌈

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

[E2E0125] ⌈

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

7.4.1 E2E_P02Protect

The E2E_P02Protect() function of E2E Profile 2 is called by a SW-C in order to protect its application data against the failure modes as shown in Table 7-5. E2E_P02Protect() therefore calculates the Counter and the CRC and puts it into the data buffer (Data). A flow chart with the visual description of the function E2E_P02Protect() is depicted in Figure 7-8 and Figure 7-9.

[E2E0126] ⌈

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

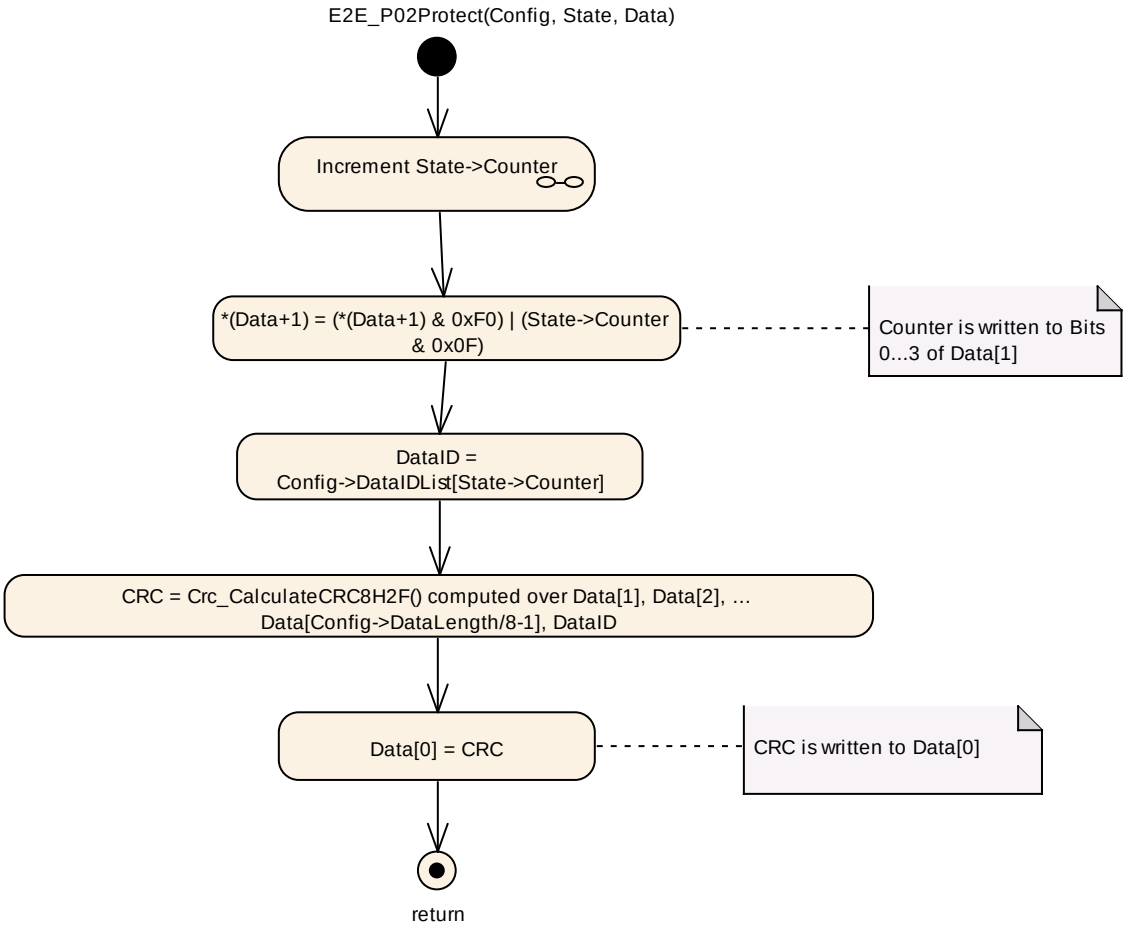


Figure 7-8: E2E_P02Protect()

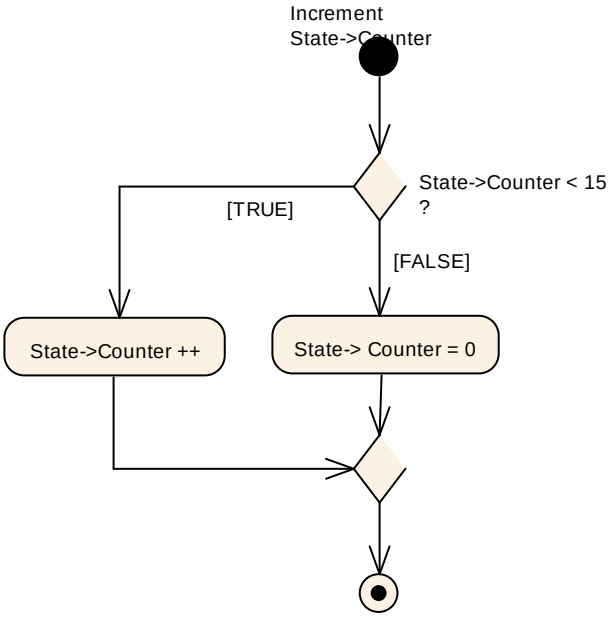


Figure 7-9: Increment Counter

[E2E0127] ⌈
In E2E Profile 2, the E2E_P02Protect() function shall increment the Counter of the state (P02SenderStateType) by 1 on every transmission request from the sending SW-C, i.e. on every call of E2E_P02Protect().⌋ ( )

[E2E0128] ⌈
In E2E Profile 2, the range of the value of the Counter shall be [0…15]. ⌋ ( )

[E2E0129] ⌈
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. ⌋ ( )

[E2E0130] ⌈
In E2E Profile 2, the E2E_P02Protect() function shall update the Counter (i.e. low nibble (Bit 0…Bit 3) of Databyte 1) in the data buffer (Data) after incrementing the Counter. ⌋ ( )

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

[E2E0132] ⌈
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. ⌋ ( )

[E2E0133] ⌈
In E2E Profile 2, the E2E_P02Protect() function shall update the CRC (i.e. Data[0]) in the data buffer (Data) after computing the CRC. ⌋ ( )

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

7.4.2 E2E_P02Check

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

A flow chart with the visual description of the function E2E_P02Check() is depicted in Figure 7-10, Figure 7-11 and Figure 7-12.

[E2E0134] ⌈
In E2E Profile 2, the E2E_P02Check() function shall perform the activities as specified in Figure 7-10, Figure 7-11 and Figure 7-12. ⌋( )

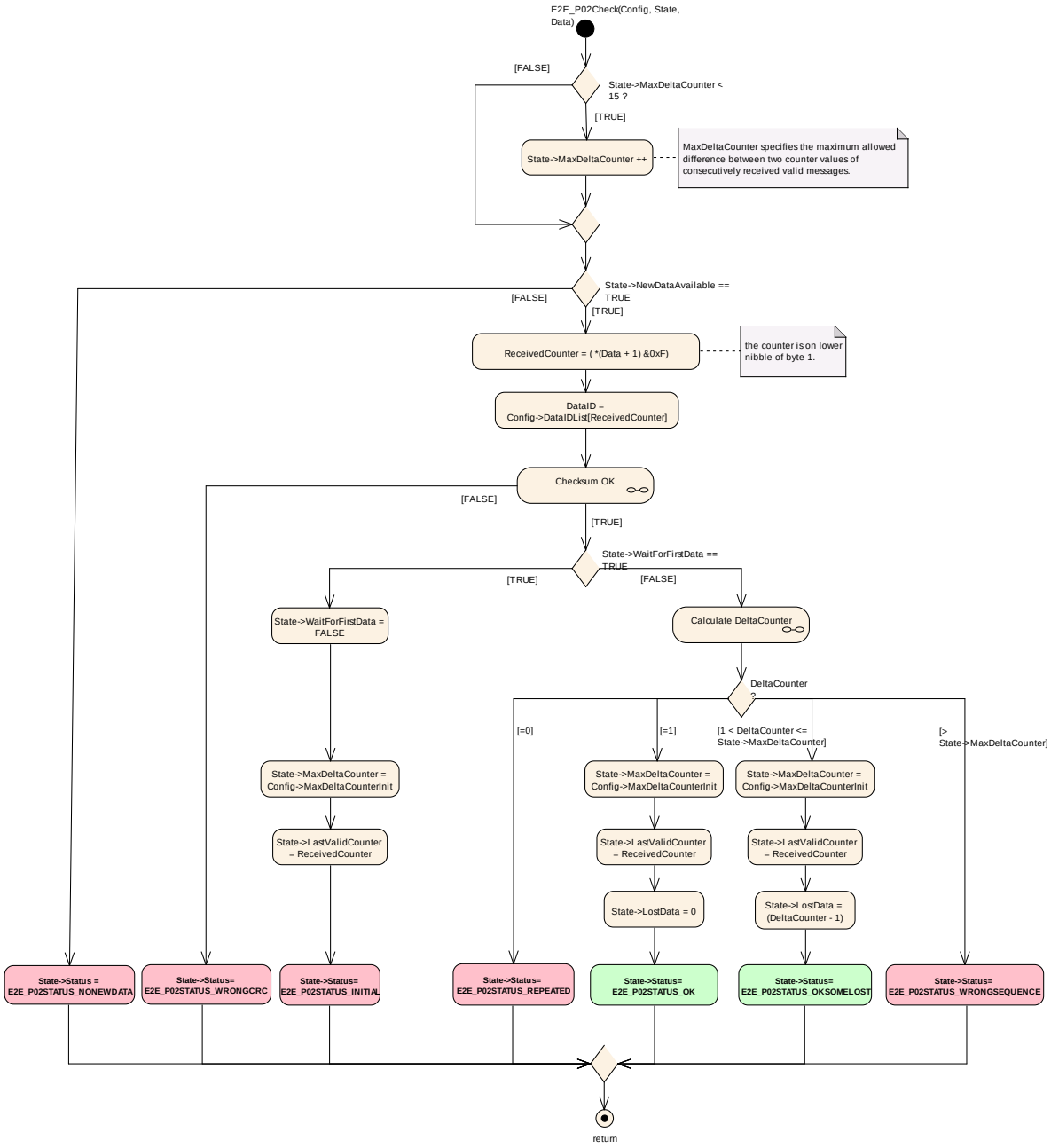


Figure 7-10: E2E_P02Check()

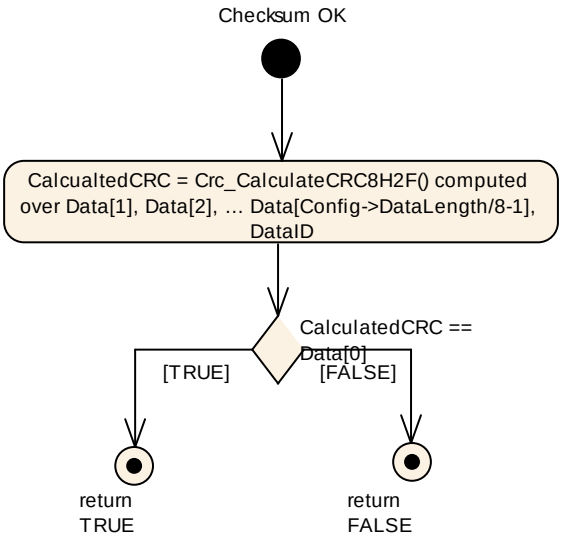


Figure 7-11: Checksum OK

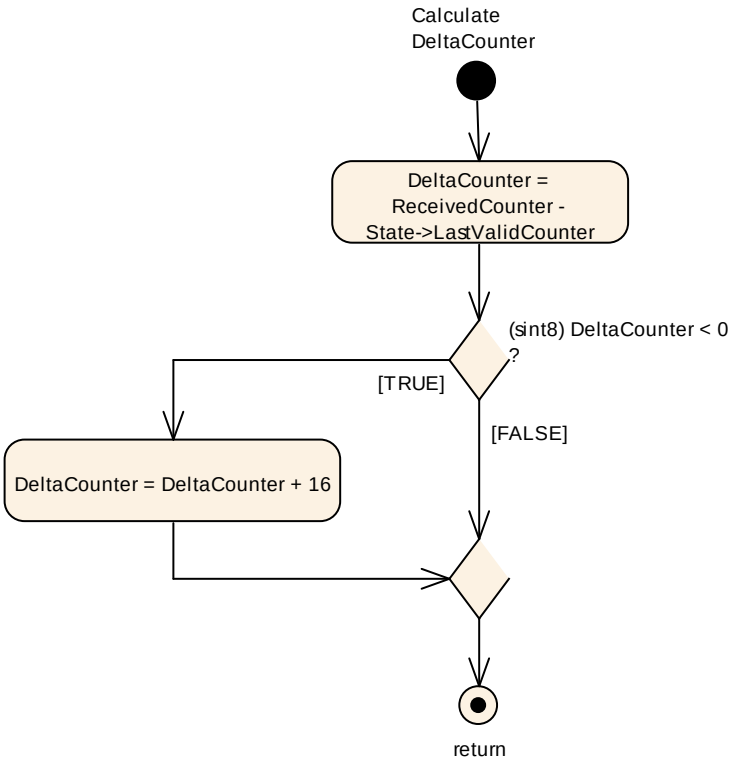


Figure 7-12: Calculate DeltaCounter

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.

[E2E0135] ⌈

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

Details on the calculation of DeltaCounter are depicted in Figure 7-12.

[E2E0136] ⌈
In E2E Profile 2, MaxDeltaCounter shall specify the maximum allowed difference between two Counter values of two consecutively received valid messages. ⌋ ( )

[E2E0137] ⌈
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). ⌋ ( )

[E2E0138] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_NONEWDATA if the attribute NewDataAvailable is FALSE. ⌋ ( )

[E2E0139] ⌈
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. ⌋ ( )

[E2E0140] ⌈
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. ⌋ ( )

[E2E0141] ⌈
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. ⌋ ( )

[E2E0142] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_INITIAL if the flag WaitForFirstData is TRUE. ⌋ ( )

[E2E0143] ⌈
In E2E Profile 2, the E2E_P02Check() function shall clear the flag WaitForFirstData if it returns with Status E2E_P02STATUS_INITIAL. ⌋ ( )

For the first message after startup 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.

[E2E0145] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_WRONGSEQUENCE if the calculated value of DeltaCounter exceeds the value of MaxDeltaCounter. ⌋ ( )

[E2E0146] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_REPEATED if the calculated DeltaCounter equals 0. ⌋ ( )

[E2E0147] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_OK if the calculated DeltaCounter equals 1. ⌋ ( )

[E2E0148] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_OKSOMELOST if the calculated DeltaCounter is greater-than 1 but less-than or equal MaxDeltaCounter (i.e. 1 < DeltaCounter ≤ MaxDeltaCounter). ⌋ ( )

[E2E0149] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set the value LostData to (DeltaCounter – 1) if it returns with the Status E2E_P02STATUS_OK or E2E_P02STATUS_OKSOMELOST. ⌋ ( )

[E2E0150] ⌈
In E2E Profile 2, the E2E_P02Check() function shall re-initialize MaxDeltaCounter with MaxDeltaCounterInit if it returns either with Status E2E_P02STATUS_OK, E2E_P02STATUS_OKSOMELOST or E2E_P02STATUS_INITIAL. ⌋ ( )

[E2E0151] ⌈
In E2E Profile 2, the E2E_P02Check() function shall set LastValidCounter to Counter of Data if it returns either with Status E2E_P02STATUS_OK, E2E_P02STATUS_OKSOMELOST or E2E_P02STATUS_INITIAL. ⌋ ( )

7.5 Version Check

[E2E0287] ⌈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 [E2E0038] for published information).

⌋ ( )

8 API specification

This chapter specifies the API of E2E Library.

8.1 Imported types

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

[E2E0017] ⌈

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

⌋ ( )

8.2 Type definitions

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

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

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

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

Example 1 - Counter offset = 8 on MSB microcontroller:

	MSB							LSB
Data[0]	7	6	5	4	3	2	1	0
	CRC with offset 0							
Data[1]	15	14	13	12	11	10	9	8
	(User data)				Counter with offset 8			

8.2.1 E2E Profile 1 types

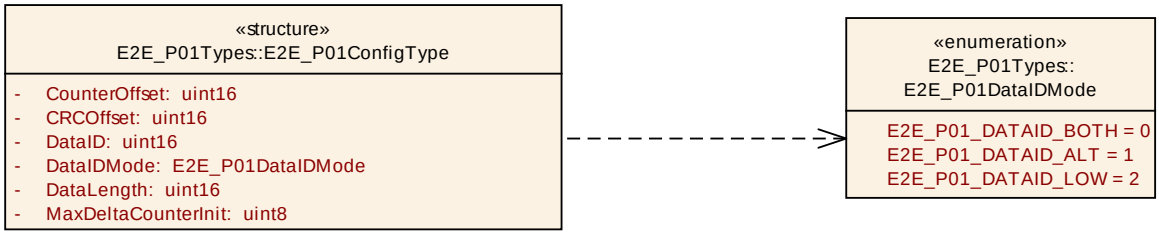


Figure 8-1: E2E Profile 1 configuration

8.2.1.1 E2E_P01ConfigType

[E2E0018] ⌈

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 multiplicity 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 multiplicity of 8.
	`uint16`	`DataID`	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in safety manual.
	`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 multiplicity of 8 and shall be ≤ 240.
	`uint8`	`MaxDeltaCounterInit`	Inital 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.
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.		

⌋ ( ) **fixme**

8.2.1.2 E2E_P01DataIDMode

[E2E0200] ⌈

Name:	E2E_P01DataIDMode	
Type:	Enumeration	
Range:	E2E_P01_DATAID_BOTH	Two bytes are included in the CRC (double ID configuration) This is used in variant 1A.
	E2E_P01_DATAID_ALT	One of the two bytes byte is included, alternating high and low byte, depending on parity of the counter (alternating ID configuration). For even counter low byte is included; For odd counters the high byte is included. This is used in variant 1B.
	E2E_P01_DATAID_LOW	Only low byte is included, high byte is never used. This is applicable if the IDs in a particular system are 8 bits.
Description:	The Data ID is two bytes long in E2E Profile 1. There are three inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.	

⌋ ( )

8.2.1.3 E2E_P01SenderStateType

[E2E0020] ⌈

Name:	E2E_P01SenderStateType		
Type:	Structure		
Element:	uint8	Counter	Counter to be used for protecting the next Data. The initial value is 0, which means that the first Data will have the counter 0. After the protection by the Counter, the Counter is incremented modulo 0xF. The value 0xF is skipped (after 0xE the next is 0x0), as 0xF value represents the error value. The four high bits are always 0.
Description:	State of the sender for a Data protected with E2E Profile 1.		

⌋ ( )

8.2.1.4 E2E_P01ReceiverStateType

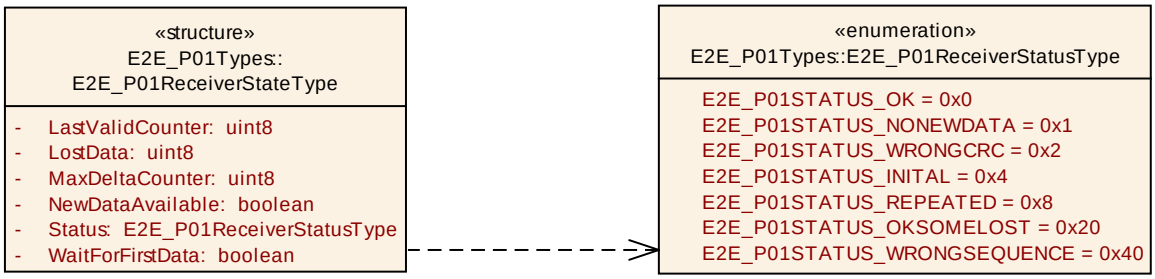


Figure 8-2: E2E Profile 1 receiver state

[E2E0021] ⌈

Name:	E2E_P01ReceiverStateType		
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_P01ReceiverStatusType	Status	Result of the verification of the Data, determined by the Check function.
Description:	State of the receiver for a Data protected with E2E Profile 1.		

⌋ ( )

8.2.1.5 E2E_P01ReceiverStatusType

[E2E0022] ⌈

Name:	E2E_P01ReceiverStatusType	
Type:	Enumeration	
Range:	E2E_P01STATUS_OK	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P01STATUS_NONEWDATA	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of

		Data have been consequently executed.
	E2E_P01STATUS_WRONGCRC	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P01STATUS_INITAL	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.
	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.	

⌋(BSW08534)

8.2.2 E2E Profile 2 types

8.2.2.1 E2E_P02ConfigType

[E2E0152] ⌈

Name:	E2E_P02ConfigType		
Type:	Structure		
Element:	uint16	DataLength	Length of Data, in bits. The value shall be a multiplicity of 8.
	uint8[16]	DataIDList	An array of appropriately chosen Data IDs for protection against masquerading.
	uint8	MaxDeltaCounterInit	Inital 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.
Description:	Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 2. The position of the counter and CRC is not configurable in profile 2.		

⌋ ( )

8.2.2.2 E2E_P02SenderStateType

[E2E0153] ⌈

Name:	`E2E_P02SenderStateType`		
Type:	`Structure`		
Element:	`uint8`	`Counter`	Counter to be used for protecting the Data. The initial value is 0, which means that the first Data will have the counter 0. After the protection by the counter, the counter is incremented modulo 16.
Description:	State of the sender for a Data protected with E2E Profile 2.		

⌋ ( )

8.2.2.3 E2E_P02ReceiverStateType

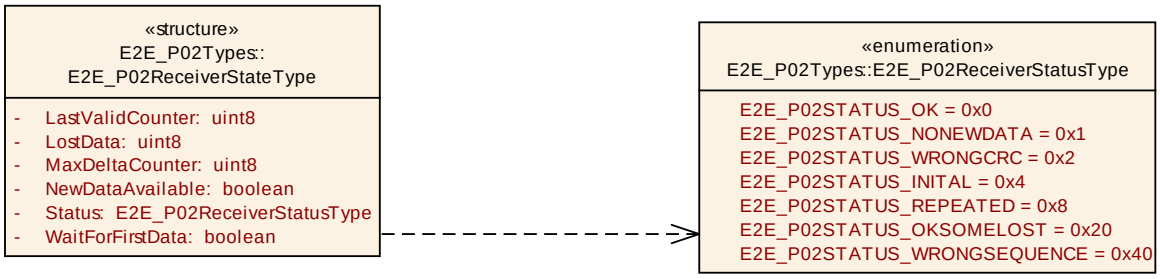


Figure 8-3: E2E Profile 2 receiver state

[E2E0154] ⌈

Name:	`E2E_P02ReceiverStateType`		
Type:	`Structure`		
Element:	`uint8`	`LastValidCounter`	Counter of last valid received message.
	`uint8`	`MaxDeltaCounter`	MaxDeltaCounter specifies

			the maximum allowed difference between two counter values of consecutively received valid messages.
	boolean	WaitForFirstData	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	boolean	NewDataAvailable	Indicates to E2E Library that a new data is available for Library to be checked. This attribute is set by the E2E Library caller, and not by the E2E Library.
	uint8	LostData	Number of data (messages) lost since reception of last valid one.
	E2E_P02ReceiverStatusType	Status	Result of the verification of the Data, determined by the Check function.
Description:	State of the sender for a Data protected with E2E Profile 2.		

⌋ ( )

8.2.2.4 E2E_P02ReceiverStatusType

[E2E0214] ⌈

Name:	E2E_P02ReceiverStatusType	
Type:	Enumeration	
Range:	E2E_P02STATUS_OK	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P02STATUS_NONEWDATA	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P02STATUS_WRONGCRC	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P02STATUS_INITAL	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.
	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.	

⌋(BSW08534)

8.3 Routine definitions

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

1. Functions
2. Inline functions
3. Macros

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

8.3.1 E2E Profile 1 routines

8.3.1.1 E2E_P01Protect

[E2E0166] ⌈

Service name:	E2E_P01Protect

Syntax:	`Std_ReturnType E2E_P01Protect(`
	`  E2E_P01ConfigType* Config,`
	`  E2E_P01SenderStateType* State,`
	`  uint8* Data`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Config	Pointer to static configuration.
Parameters (inout):	State	Pointer to port/data communication state.
	Data	Pointer to Data to be transmitted.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see E2E0047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID.	

⌋ ( )

8.3.1.2 E2E_P01Check

[E2E0158] ⌈

Service name:	E2E_P01Check	

Syntax:	`Std_ReturnType E2E_P01Check(`
	`  E2E_P01ConfigType* Config,`
	`  E2E_P01ReceiverStateType* State,`
	`  uint8* Data`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Config	Pointer to static configuration.
	Data	Pointer to received data.
Parameters (inout):	State	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see E2E0047.
Description:	Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.	

⌋ ( )

8.3.2 E2E Profile 2 routines

8.3.2.1 E2E_P02Protect

[E2E0160] ⌈

Service name:	E2E_P02Protect	

Syntax:	`Std_ReturnType E2E_P02Protect(`
	`  E2E_P02ConfigType* Config,`
	`  E2E_P02SenderStateType* State,`
	`  uint8* Data`
	`)`	

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

⌋ ( )

8.3.2.2 E2E_P02Check

[E2E0161] ⌈

Service name:	E2E_P02Check	

Syntax:	`Std_ReturnType E2E_P02Check(`
	`  E2E_P02ConfigType* Config,`
	`  E2E_P02ReceiverStateType* State,`
	`  uint8* Data`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Config	Pointer to static configuration.
	Data	--
Parameters (inout):	State	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR

		E2E_E_OK For definitions for return values, see E2E0047.
Description:	Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.	

⌋ ( )

8.3.3 Elementary protocol routines

The E2E Library provides various elementary functions enabling to build custom E2E profiles. First, it provides a couple of CRC routines, which are just wrappers above CRC library CRC8 functions, for computing CRC8 over multi-byte integers and for computing CRC8 over arrays of multibyte 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.

[E2E0106] ⌈
When calculating a CRC8 by calling any E2E_CRC8_* function (single call), the caller shall use the protocol-specific start value as E2E_StartValue. ⌋ ( )

[E2E0107] ⌈
When calculating a resulting CRC by multiple calls, the first call shall use 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. ⌋ ( )

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

8.3.3.1 E2E_CRC8<InTypeMn>

[E2E0092] ⌈

Service name:	E2E_CRC8<InTypeMn>	

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 transmited 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, but it makes the API more systematic. Relation to Crc_CalculateCRC8():E2E_CRC8_*() may simply call Crc_CalculateCRC8() is a loop (this is a recommendation for implementation). The function uses SAE J1850 polynomial, but with 0x00 as start value and XOR value.	

⌋ ( )

[E2E0091] ⌈

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

⌋ ( )

8.3.3.2 E2E_CRC8<InTypeMn>Array

[E2E0094] ⌈

Service name:	E2E_CRC8<InTypeMn>Array	

Syntax:	`uint8 E2E_CRC8<InTypeMn>Array(`
	`  uint16 Length,`
	`  <InType>* Data,`
	`  uint8 StartValue`
	`)`	

Service ID[hex]:	0x0A, 0x0B, 0x0C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Length	Length of array (data block) to be calculated in bytes.
	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0x00, or (2) 0x00 if it is the first run.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	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, 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. (this is a recommendation for implementation). The function uses SAE J1850 polynomial, but with 0x00 as start value and XOR value.	

⌋ ( )

[E2E0095] ⌈

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

⌋ ( )

8.3.3.3 E2E_CRC8H2F<InTypeMn>

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

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

[E2E0096] ⌈

Service name:	E2E_CRC8H2F<InTypeMn>	

Syntax:	`uint8 E2E_CRC8H2F<InTypeMn>(`
	`  <InType> Data,`
	`  uint8 StartValue`
	`)`	

Service ID[hex]:	0x0D, 0x0E, 0x0F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0xFF, or (2) 0xFF if it is the first run.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	CRC8 value calculated based on the CRC from previous iteration and over a primitive data element from the current iteration, using CRC8 polynomial 0x2F.
Description:	InTypeMn: {u8, u16, u32}, which is the one corresponding to InType. Utility function for calculating CRC over primitive data types transmited 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, but it makes the API more systematic. The function uses not the SAE polynomial, but 0x2F.	

⌋ ( )

[E2E0276] ⌈

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

⌋ ( )

8.3.3.4 E2E_CRC8H2F<InTypeMn>Array

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

[E2E0097] ⌈

Service name:	E2E_CRC8H2F<InTypeMn>Array	

Syntax:	`uint8 E2E_CRC8H2F<InTypeMn>Array(`
	`  uint16 Length,`
	`  <InType>* Data,`
	`  uint8 StartValue`
	`)`	

Service ID[hex]:	0x10, 0x11, 0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Length	Length of array (data block) to be calculated in bytes.
	Data	Current value over which the CRC is to be computed. InType: {uint8, uint16, uint32}
	StartValue	(1) CRC value from the previous iteration XORed with 0xFF, or (2) 0xFF if it is the first run.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	CRC8 value calculated based on the CRC from previous iteration and over the 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, but it makes the API more systematic. The function uses not the SAE polynomial, but 0x2F.	

⌋ ( )

[E2E0098] ⌈

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

⌋ ( )

8.3.3.5 E2E_UpdateCounter

[E2E0099] ⌈

Service name:	E2E_UpdateCounter	

Syntax:	`uint8 E2E_UpdateCounter(`
	`  uint8 Counter`
	`)`	

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

⌋ ( )

8.3.4 Auxiliary Functions

8.3.4.1 E2E_GetVersionInfo

[E2E0032] ⌈

Service name:	E2E_GetVersionInfo	

Syntax:	`void E2E_GetVersionInfo(`
	`  Std_VersionInfoType* VersionInfo`
	`)`	

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

⌋ (BSW003)

[E2E0033] ⌈
The function E2E_GetVersionInfo shall return the version information of this module. The version information includes:

- vendor ID
- module ID

- sw_major_version
- sw_minor_version
- sw_patch_version⌋ ( )

8.4 Call-back notifications

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

8.5 Scheduled functions

None. The E2E library does not have scheduled functions.

8.6 Expected Interfaces

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

[E2E0110] ⌈
The E2E library shall not call any functions from external modules apart from explicitly listed expected interfaces of E2E Library. ⌋ ( )

8.6.1 Mandatory Interfaces

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

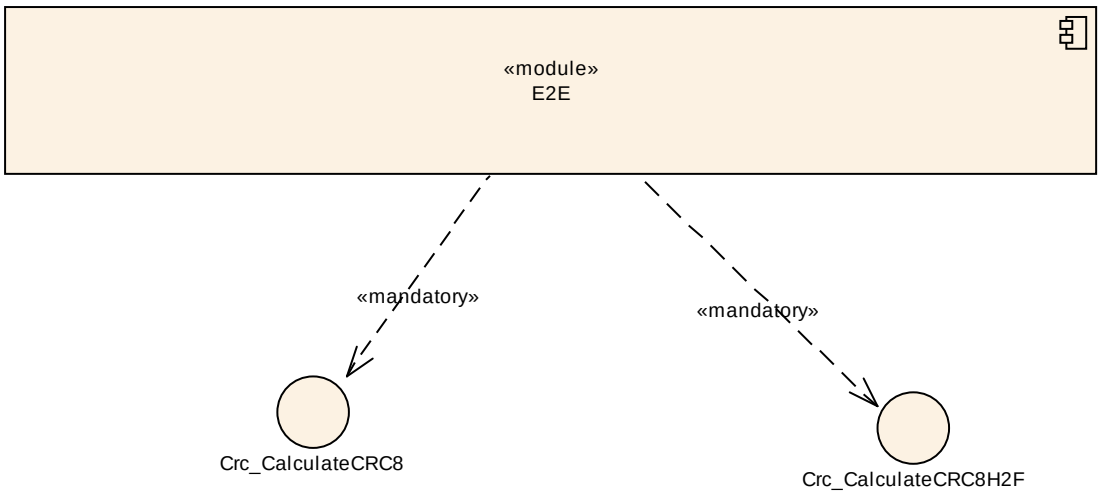


Figure 8-4: Expected mandatory interfaces by E2E library

9 Sequence Diagrams for invoking E2E Library

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

9.1 Sender

[E2EUSE202] ⌈
During its initialization, the Sender shall instantiate the structures PXXConfigType and PXXSenderStateType, separately for each Data to be protected. ⌋ ( )

[E2EUSE203] ⌈
During its initialization, the Sender shall initialize the PXXConfigType with the required configured settings, for each Data to be protected. ⌋ ( )

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

[E2EUSE204] ⌈
During its initialization, the Sender shall initialize the E2E_PXXSenderStateType for each Data, with the configured following values: Counter = 0. ⌋ ( )

[E2EUSE205] ⌈
In every send cycle, the Sender shall invoke once the function E2E_PXXProtect() and then once the function to transmit the data (e.g. Rte_Send_<p>_<o>() or PduR_ComTransmit()).

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

9.1.1 Sender of Data Elements

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

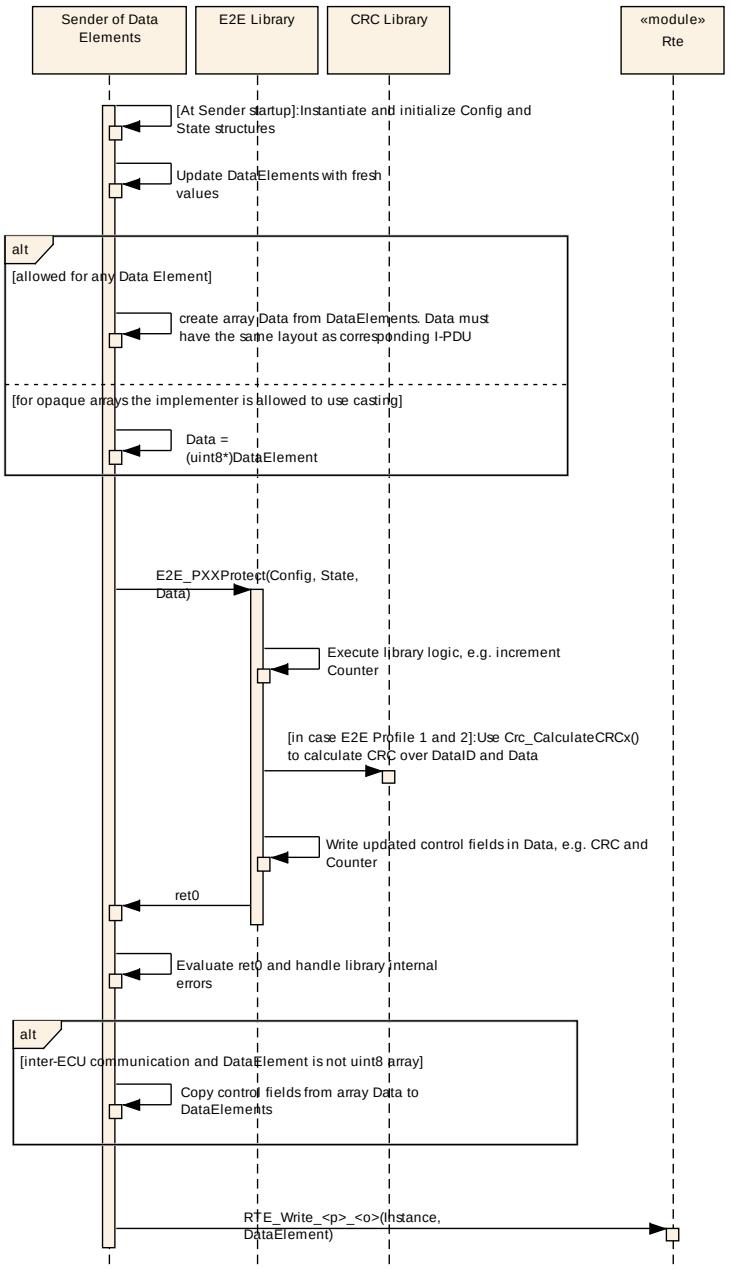


Figure 9-1: Sender of Data Elements

After the new Data Element is available, before calling E2E_PXXProtect(), the Sender of Data Elements, shall:

[E2EUSE230] ⌈
In case the Data Element communication is inter-ECU and the Data Element is not an opaque uint8 array, then the user of the E2E Library shall serialize the Data Element into the array Data. The content of the array Data shall be the equal to the content of the serialized representation of corresponding signal group in an I-PDU.

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

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

[E2EUSE232] ⌈
For sending of Data Elements different from opaque arrays, the caller of E2E Library shall serialize the Data Element to Data, then it shall call the E2E_PXXProtect() routine and then it shall copy back the control fields from Data to Data Element.

By its nature, the serialization involves data copying. If a Data Element is an opaque array, then the 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 of I-PDUs

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

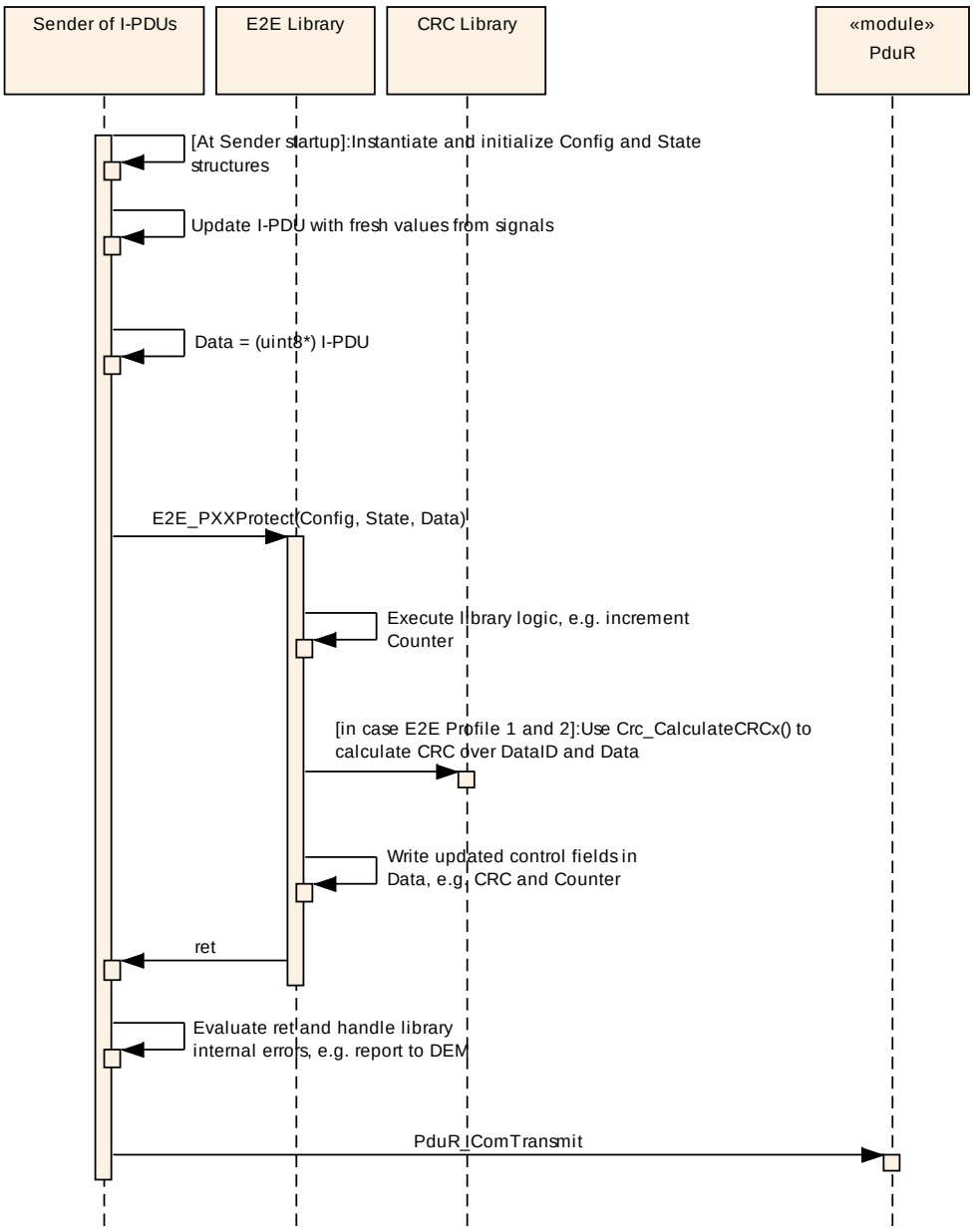


Figure 9-2: Sender of I-PDUs

9.2 Receiver

[E2EUSE206] ⌈
During its initialization, the Receiver shall instantiate the structures PXXConfigType and PXXReceiverType. ⌋ ( )

[E2EUSE207] ⌈

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

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

[E2EUSE208] ⌈
During its initialization, the Receiver shall initialize the E2E_PXXReceiverStateType with the following values:

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

[E2EUSE209] ⌈
In every receive cycle, the Receiver shall:

1. invoke once the reception function and a function to check if new data is available (e.g. by calling optionally the function Rte_IsUpdated_<p>_<o>() and then by calling function Rte_Read_<p>_<o>()),

2. Set the attribute State->NewDataAvailable to TRUE if new data has been received without any errors (e.g.:
NewDataAvailable = ((IsUpdated == TRUE) && (Rte_Read_Status == RTE_E_OK)))

3. Update Data, using received Data Element or I-PDU.

4. Call once the function E2E_PXXCheck().

5. Handle results (return value and State parameter) returned by E2E_PXXCheck().

The Functions E2E_PXXCheck() return the results of verification, by means of parameter State. Within the State (structure E2E_PXXReceiverStateType), there is the attribute LostData, which is has a defined value and makes sense only for the following states: E2E_PXXSTATUS_OK and E2E_PXXSTATUS_OKSOMELOST. ⌋( )

[E2EUSE233] ⌈
If the return from the function E2E_PXXCheck() is different than E2E_PXXSTATUS_OK and E2E_PXXSTATUS_OKSOMELOST, then the caller shall not evaluate the attribute State->LostData. ⌋( )

9.2.1 Receiver of Data Elements

The diagram below species the overall sequence involving the E2E Library called by the Receiver 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.

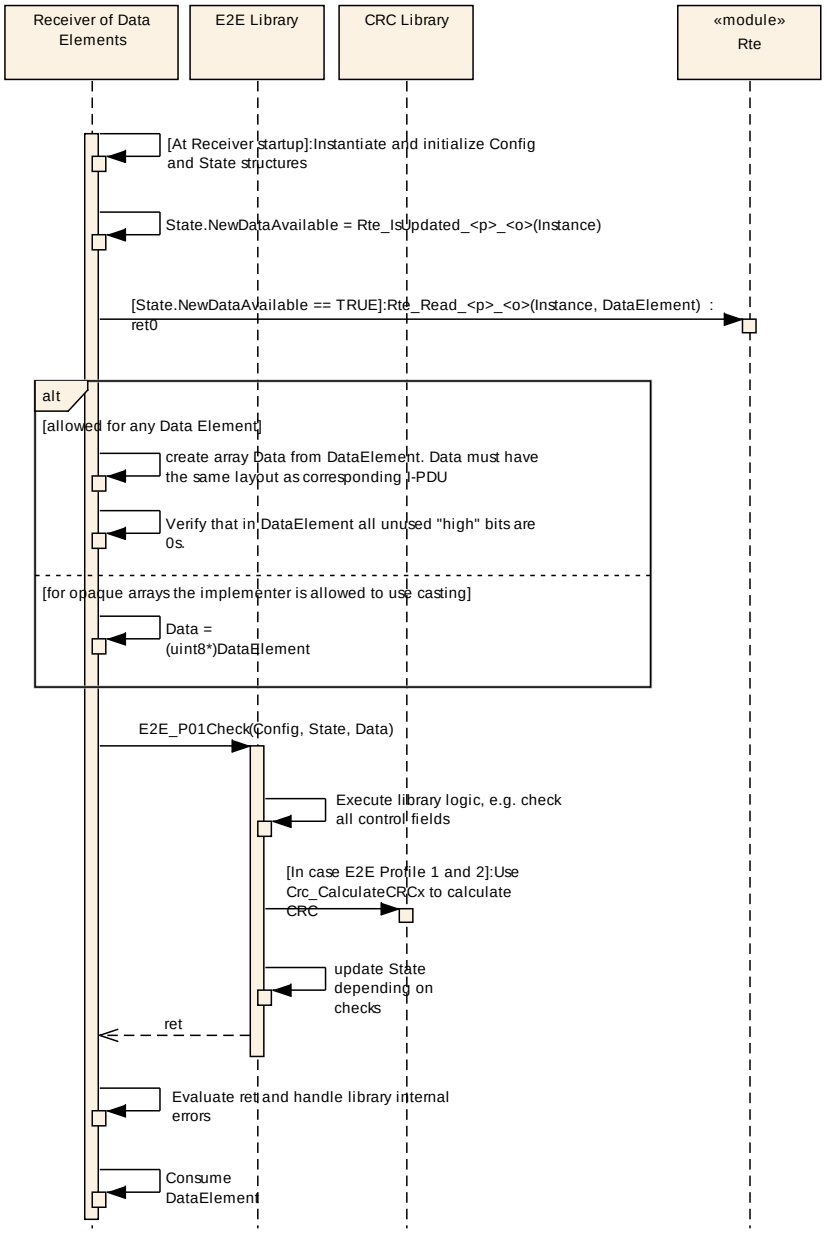


Figure 9-3: Receiver of Data Elements

[E2EUSE0277] ⌈

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

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

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

[E2EUSE0278] 「
For reception of Data Elements different from opaque arrays, the caller of E2E Library shall serialize the Data Element to Data, then it shall call the check routine.

9.2.2 Receiver of I-PDUs

The diagram below summarizes the sequence involving the E2E Library by the Receiver of I-PDUs.

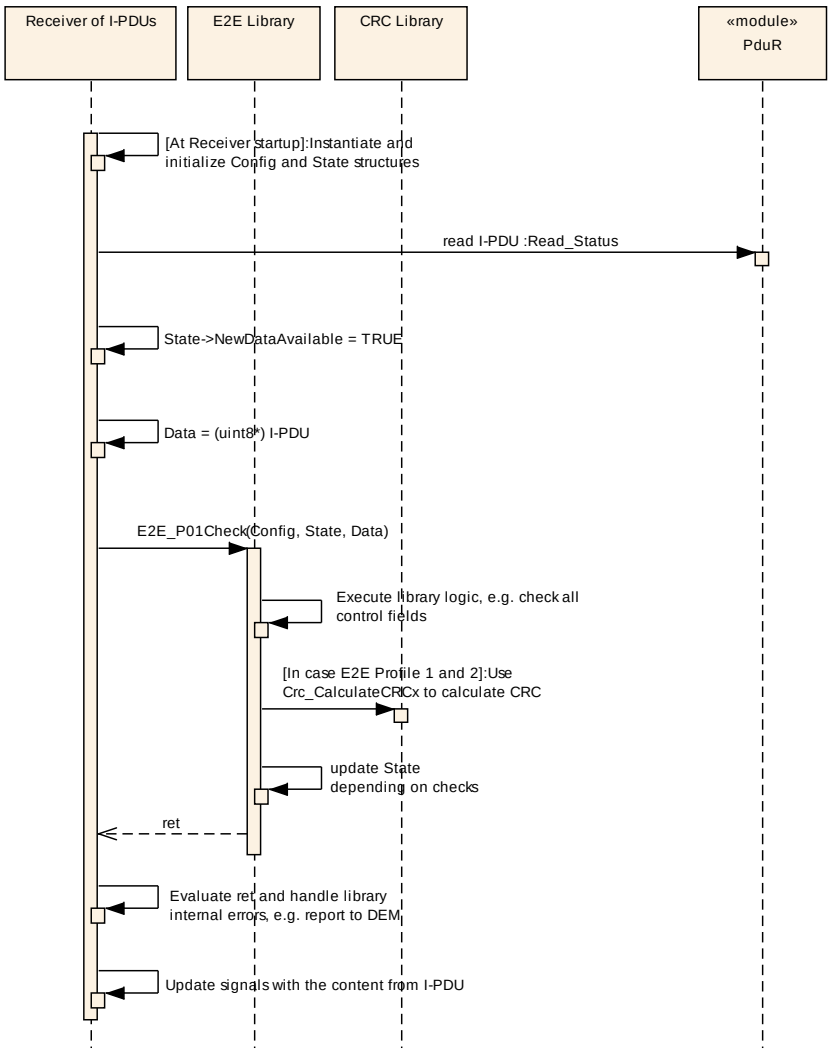


Figure 9-4: Receiver of I-PDUs

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.

[E2E0037] ⌈
The E2E library shall not have any configuration options. ⌋ (BSW00344, BSW00345, BSW159, BSW167, BSW171, BSW170, BSW101)

10.1 Published Information

[E2E0038] ⌈
The standardized common published parameters as required by BSW00402 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]. ⌋ (BSW004)

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

11 Annex A: Safety Manual for usage of E2E Library

This chapter contains application hints to enable an appropriate 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.

Additionally, E2E Profile 1 has two standard variants. E2E Profile 2 has less configuration options and does not have variants.

[E2EUSE053] ⌈
Any user of E2E Library shall use whenever possible the *standard E2E Profile variants* of E2E Profiles, and apply non-standard E2E Profile configurations only for specific cases (like legacy applications). ⌋ ( )

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.

[E2EUSE235] ⌈
The user (caller) of E2E Library, in particular the receiver, shall provide the error handling mechanisms for the faults detected by the E2E Library. ⌋ ( )

11.3 Maximal lengths of Data, communication buses

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

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

E2E Profile	Max applicable length including control fields for inter-ECU communication
E2E Profile 1	30
E2E Profile 2	42

Figure 11-1: Maximum lengths

[E2EUSE051] ⌈
In case of intra-ECU communication over FlexRay or CAN, the length of the complete Data (including application data, CRC and counter) protected by E2E Profile 1 should not exceed 30 bytes. ⌋ ( )

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

[E2EUSE061] ⌈
In case of CAN or LIN the length of the complete data element (including application data, CRC and counter) protected by E2E Profile 1 shall not exceed 8 bytes. ⌋ ( )

The requirements E2EUSE051 and E2EUSE061 only contain 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.

[E2EUSE236] ⌈
When using E2E Library, the designer of the functional or technical safety concept of a particular system using E2E Library shall evaluate the maximum permitted length of the protected Data in that system, to ensure an appropriate error detection capability. ⌋ ( )

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

[E2EUSE170] ⌈
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 handle the reception of one erroneous data element, which error was not detected by the E2E library. What is *not* required is that an SW-C tolerates two consecutive undetected erroneous data elements, because it

is enough unlikely that two consecutive Data are wrong AND that for both Data the error remains undetected by the E2E library. ⌋( )

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

11.4 Methodology of usage of E2E Library

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

In the first step, the user selects the architectural approach how E2E Library is used in a given system (through COM callouts, through E2E Protection wrapper etc). There are several architectural solutions of usage of E2E Library described in Chapter 12.

In the second step, the user selects which Data Elements or I-PDUs 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 I-PDU 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 I-PDU 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 I-PDUs), no two different data elements (or I-PDUs) are allowed to have the same Data ID (E2E Profiles 1 and 3) or same DataIDList[] (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.

[E2EUSE071] ⌈
Any user of E2E Library shall ensure, that within one implementation of a communication network every safety-related Data Element, protected by E2E Library, has a unique Data ID (see Profile 1 and 3) or a unique DataIDList[] (see Profile 2). ⌋ ( )

[E2EUSE237] ⌈
Any user of E2E Library shall ensure, that within one implementation of a communication network every safety-related I-PDU, protected by E2E Library, has a unique Data ID (see Profile 1 and 3) or a unique DataIDList[] (see Profile 2). ⌋ ( )

Note: For Profile 1 requirement (E2EUSE071) may not be sufficient in some cases, because Data ID is longer than CRC, which results with additional requirements E2EUSE072 and E2EUSE073. 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). For both cases, 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 standard variant 1A, and all 16 bits are always included in the CRC calculation. This results in the fact that 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 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.

[E2EUSE072] ⌈
Any user of Profile 1 shall ensure, that assuming two data elements DE1 and DE2, on the same system (vehicle): for any data element DE1 having ASIL B, ASIL C or ASIL D requirements with Data ID DI1, there shall not exist any other Data Element DE2 (of any ASIL) with Data ID DI2, where:
Crc_CalculateCRC8(start value: 0x00, data[2]: {lowbyte (DI1),highbyte(DI1)})
=
Crc_CalculateCRC8(start value: 0x00, data[2]: {lowbyte (DI2),highbyte(DI2)}).
⌋ ( )
The above requirement limits the usage of Data IDs of data having ASIL B, C, D to 255 distinct values in a given ECU, but gives the flexibility to define the Data IDs within the 16-bit naming space.

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

[E2EUSE073] ⌈
Any user of Profiles 1 shall ensure, that assuming two data elements DE1 and DE2, on the same system (vehicle): for any data element DE1 having ASIL A requirements with Data ID DI1, there shall not exist any other Data Element DE2 (having ASIL A requirements) with Data ID DI2 and of the same length AS DE1, where
Crc_CalculateCRC8(start value: 0x00, data[2]: {lowbyte (DI1),highbyte(DI1)})
=
Crc_CalculateCRC8(start value: 0x00, data[2]: {lowbyte (DI2),highbyte(DI2)}).
⌋ ( )
The above two requirements E2EUSE072 and E2EUSE073 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.

11.6 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.

[E2EUSE0259] ⌈
Any developer of a custom-built E2E Profile using elementary mechanisms provided by E2E Library shall ensure that this custom built E2E Profile is adequate for safety-related communications within the automotive domain. ⌋ ( )

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

11.7 I-PDU Layout

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

[E2EUSE062] ⌈
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. ⌋ ( )

[E2EUSE063] ⌈
Signals that have length >= 8 bits should start or finish at the byte limit of an I-PDU. ⌋ ( )

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

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

.

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

Figure 11-2: Example for alignment not following recommendations

11.7.2 Unused bits

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

[E2EUSE173] ⌈
Unused bits located in a to be protected data structure containing one or more safety-related signals (e.g. signal-group or I-PDU) shall be initialized with each bit set to 1 before protecting this data structure by using E2E-Library.⌋ ( )

For example, an unused high nibble gets the value 0xF0, low nibble – 0x0F, unused byte – 0xFF.

11.7.3 Byte order (Endianness)

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

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

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

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

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

1. Sender COM converts the byte Endianness of the signals (if configured/needed),
2. Sender COM copies the converted signal on I-PDU (serializes the signal), while copying only used bits from the signals,

3. Sender COM delivers unaltered I-PDU to receiver COM (an I-PDU is just a byte array unaltered by lower layers of the network stack),

4. Receiver COM converts the Endianness of the signals in the received I-PDU (if configured). It may also do the sign extension (if configured),
5. Receiver COM returns the converted signals.

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

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

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

Table 11-1: Networks and their byte order

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

[E2EUSE055] ⌈
Any user of E2E Profile 1, 2 and 3 designing a SW-C shall place multibyte data in the data elements in Little Endian order. ⌋ ( )

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

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

[E2EUSE057] ⌈
Any user of E2E Library shall ensure that a signal/data element is either opaque or non-opaque on both sides (i.e. the sender and the receiver side).

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

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

[E2EUSE056] ⌈
Any user of E2E Library shall ensure that if an opaque data type is used as a signal/data element by a SW-C, then the signal/data element is in Little Endian order and the sender/receiver SW-C is responsible for converting the data element from/to the Little Endian order to its own byte order. ⌋ ( )

11.7.4 Bit order

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

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

At the level of software, the microcontroller bit order is not visible. For example, a software module, accessing a bit 3 (of value 2^3) does not care or know if the bit is 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
Byteflight (not supported by AUTOSAR up to Release 4.0)	MSB first

MOST (not supported by AUTOSAR up to Release 4.0)	MSB first

Table 11-2: Networks and their bit order

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

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

11.8 RTE configuration constraints for SW-C level protection

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

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

11.8.1 Communication model for SW-C level protection

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

[E2EUSE087] ⌈
In case the E2E Library is used to protect Data Elements, then the user of E2E Library shall use the Sender-receiver communication model for safety-related communication. ⌋ ( )

11.8.2 Multiplicities for SW-C level protection

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

[E2EUSE0258] ⌈
In case the E2E Library is used to protect Data Elements, then the selected multiplicity shall be 1:N or 1:1. ⌋ ( )

11.8.3 Explicit access

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

data – this is not configurable. RTE transmits the data to the receiver concurrently to the execution of the sender.

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

1. The receiver waits for new data: it is blocked/waiting until new data element from the sender arrives (RTE communication modes “wake up of wait point” and “activation of Runnable entity”)

2. The receiver gets the currently available data element from RTE, i.e. the most recent data element (RTE communication modes “Implicit data read access” and “Explicit data read access”)

As explained in 7.3.5, E2E Profile 1, 2 and 3 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.

[E2EUSE089] ⌈
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").⌋ ( )

11.9 Definition of IPDU IDs

E2E Library does not support N:1 communication, because it is important to differentiate different senders. Therefore, similarly to E2EUSE243, there is an equivalent requirement for I-PDU protection.

[E2EUSE0260] ⌈
In case the E2E Library is used to protect I-PDUs, then each such protected I-PDU shall have a unique I-PDU ID in a system. ⌋ ( )

12 Annex B: Application hints on usage of E2E Library

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

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

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

The E2E Library can be used to protect safety-related data elements exchanged between SW-Cs by means of E2E Protection Wrapper (Chapter 12.1). Second, E2E Library can be also used to protect safety-related I-PDUs (Chapter 12.2).

It is also possible to have mixed scenarios:

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

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

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

This annex describes two exemplary, basic solutions how E2E Library can be invoked. First, this is by means of a dedicated sub-layer for a SW-C or several SW-Cs (which is called E2E Protection Wrapper, see Chapter 12.1). Secondly, this can be done by means of dedicated COM Callouts invoking E2E Library to protect I-PDUs (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, the user defines EndToEnd* metaclasses and associates them to ISignalIPdu metaclass (representing the I-PDU).

[E2EUSE0271] ⌈
A given I-PDU, shall not be at the same protected by means of COM E2E callouts (through association with ISignalIPdu) and by means of E2E Protection Wrapper (through association with E2E Protection Wrapper. ⌋ ( )

12.1 E2E Protection Wrapper

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

The usage of E2E Protection Wrapper allows a use of VFB communication between SW-Cs[2], 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.

[E2EUSE0292] ⌈
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.

[2] 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 device driver.

The E2E Protection Wrapper itself is not a part of E2E Library. However, its options are standardized. Most of options for E2E Protection Wrapper are in System Template [12] and some of them are in Software Component Template [11].⌋( )

[E2EUSE0249]⌈
The integrity of operation of E2E Protection Wrapper (for transmitting/converting safety-related data) shall be guaranteed.

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

[E2E0288]⌈
Each E2E Protection Wrapper function shall not be called concurrently.⌋( )

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

12.1.1 Functional overview

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

For a Data Element to transmit, there is a pair of functions Rte_Write_<p>_<o>() and Rte_Read_<p>_<o>() generated for Sender and respectively 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/error bits.

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):

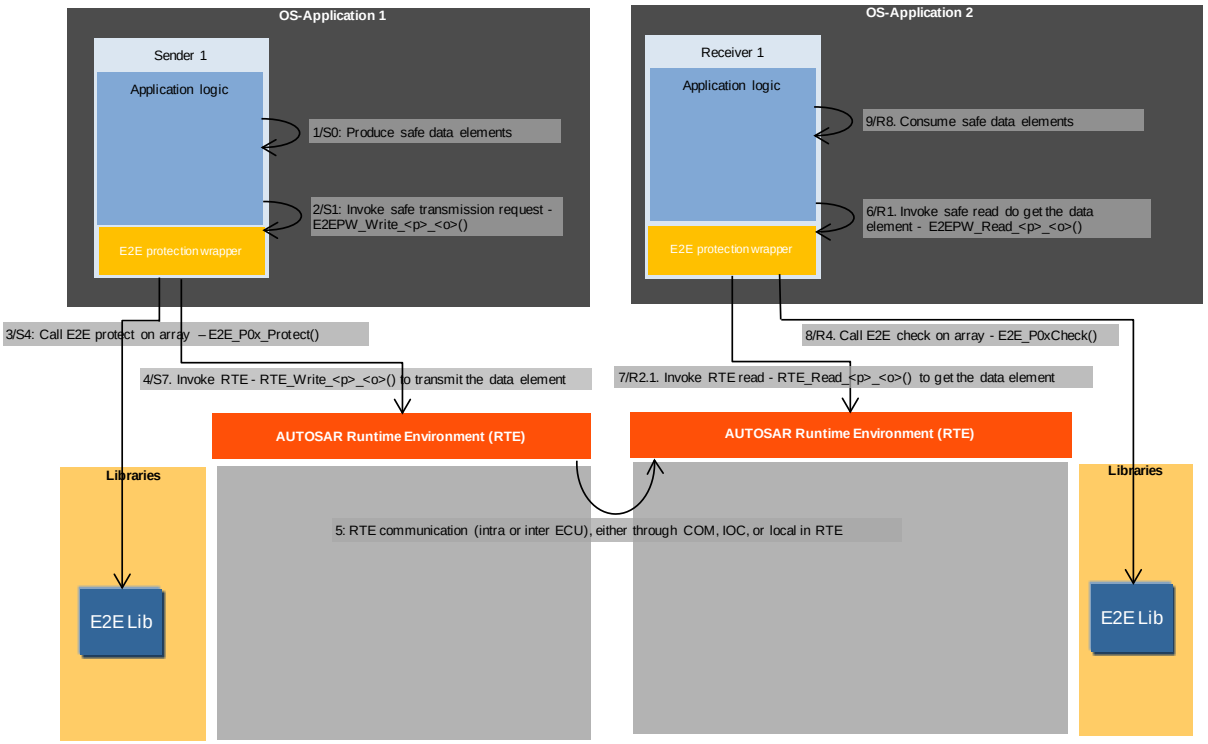


Figure 12-1: Example E2E Protection Wrapper - overall flow

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.

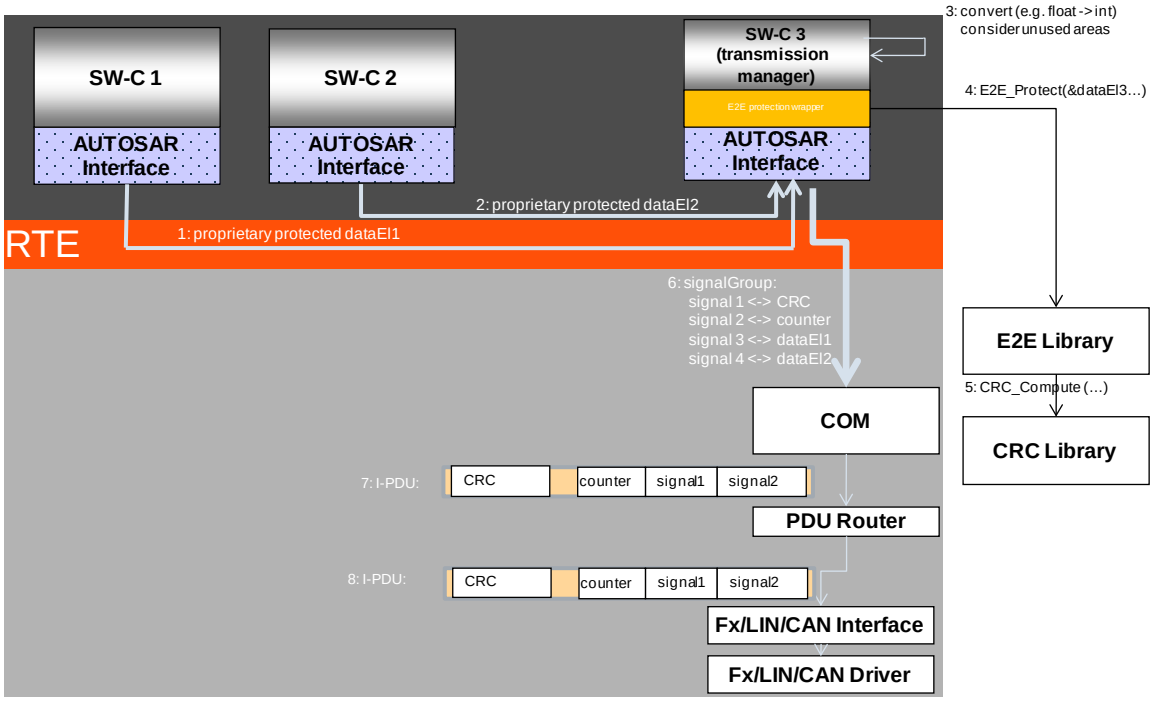


Figure 12-2: Example Transmission Manager – sender ECU

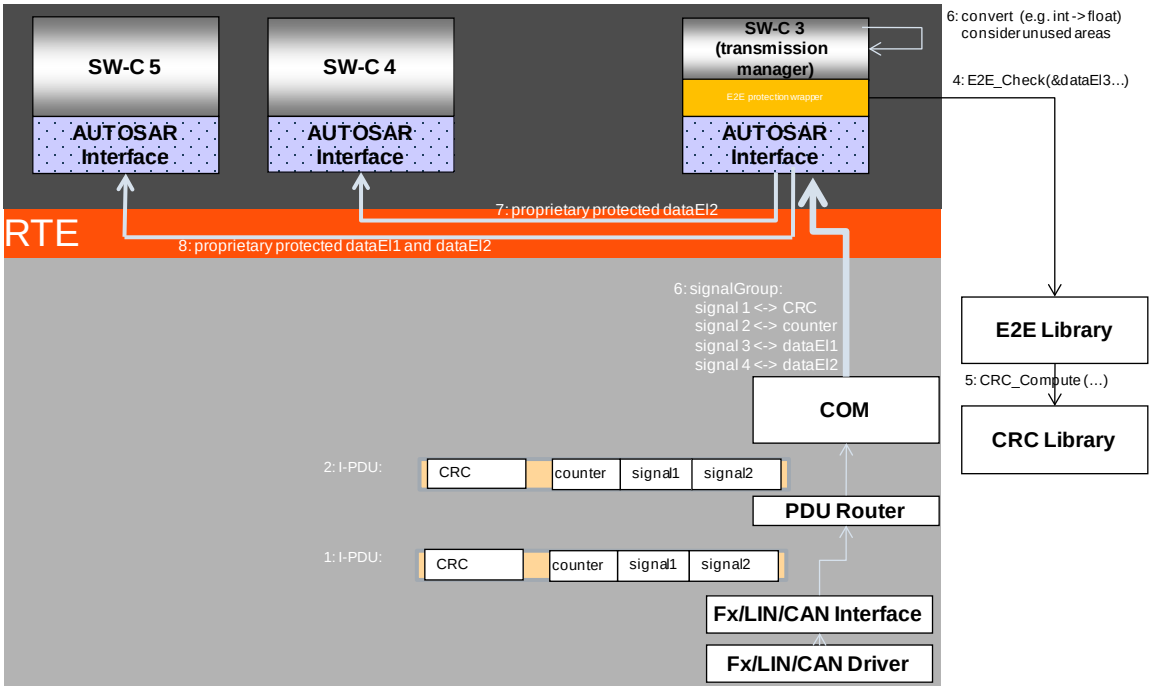


Figure 12-3: Example Transmission Manager – receiver ECU

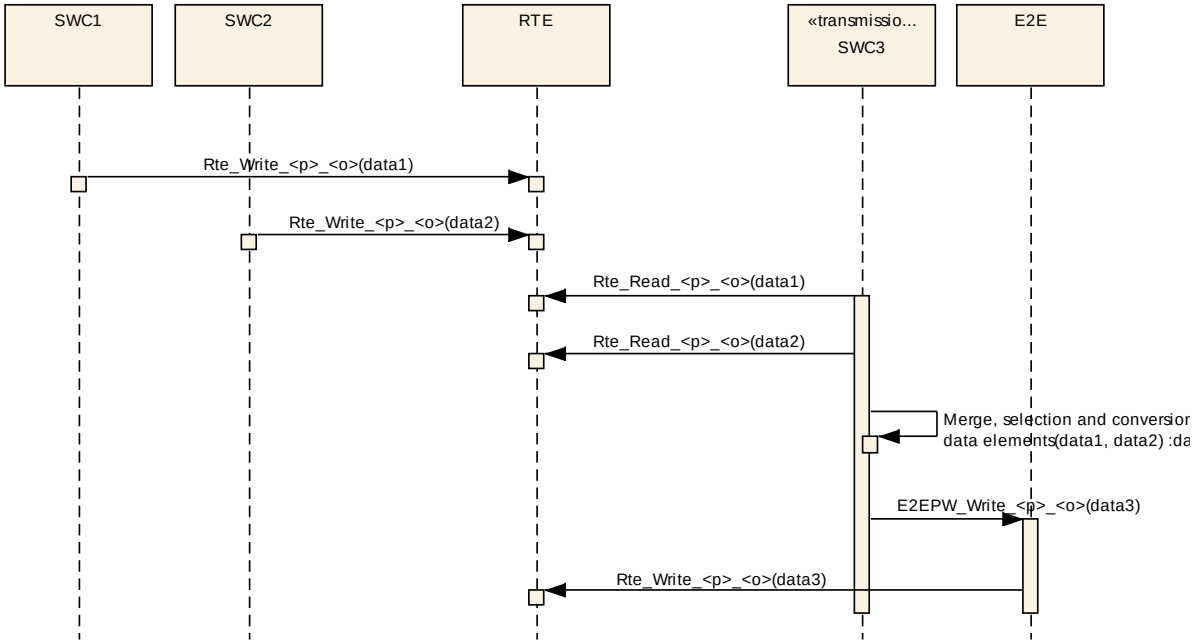


Figure 12-4: Example Transmission Manager – sender ECU sequence

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.

[E2EUSE0213] ⌈
The implementation of the Transmission Manager (as a safety-related Software Component), shall comply with the requirements for the development of safety-related software for automotive domain. ⌋ ( )

12.1.3 Application scenario with E2E Manager and Conversion Manager

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

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

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

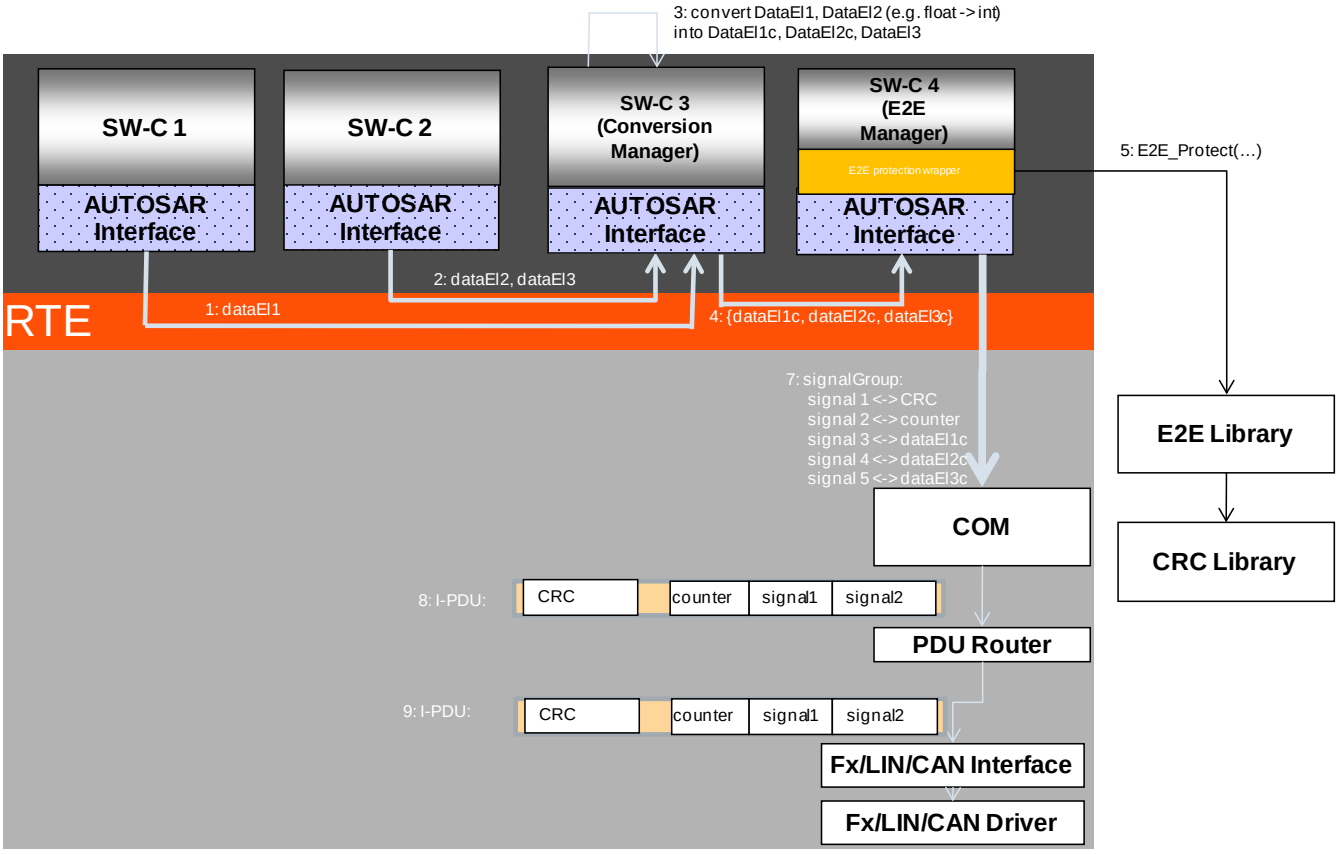


Figure 12-5: E2E Manager and Conversion Manager – sender ECU

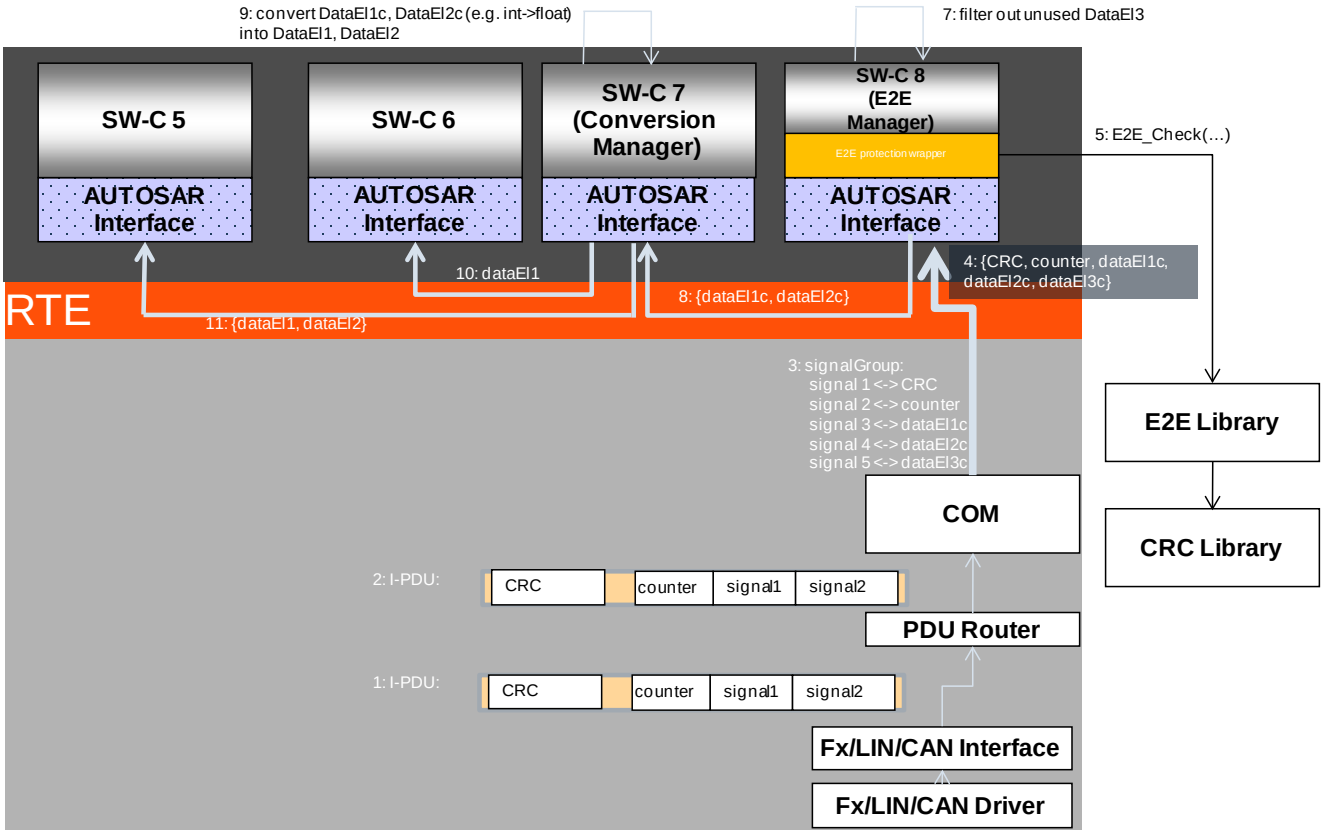


Figure 12-6: E2E Manager and Conversion Manager – receiver ECU

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:

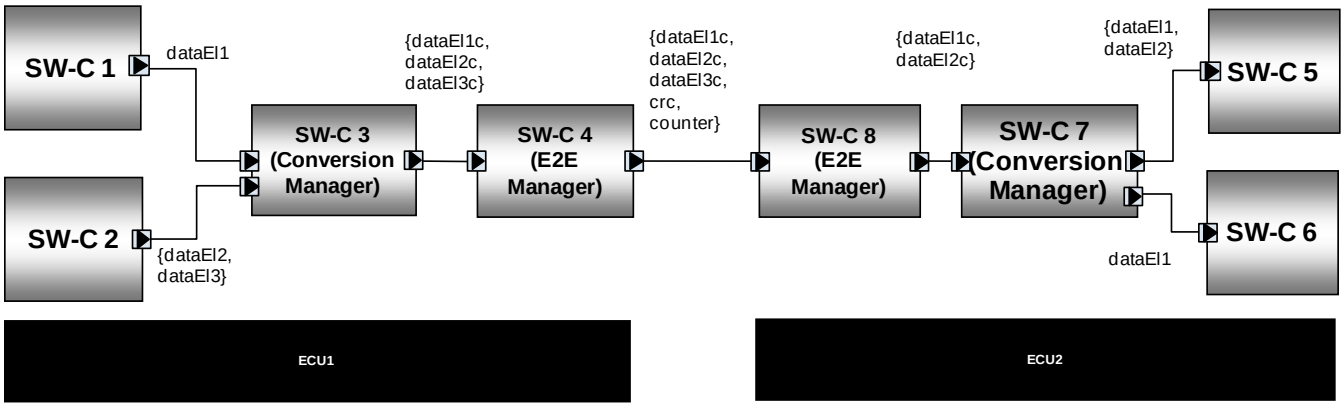


Figure 12-7: E2E Manager and Conversion Manager - system configuration

The E2E protection wrapper of E2E manager can be automatically generated, as described in 12.1.7.

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;
```

[E2EUSE0274] ⌈
E2E Manager shall have complex data elements with prefix Input or with prefix Output. There is one-to-one relationship between the data element with input prefix and data element with output prefix ⌋ ( )

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

[E2EUSE0275] ⌈
The output data element shall contain the subset of primitive data elements of those of the corresponding input data element (in particular, they may be equal). ⌋ ( )

In the example above, Outputswc8 contains the subset of attributes of Inputswc8. It does not contain 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`**.

[E2EUSE0272] ⌈
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. ⌋ ( )

[E2EUSE0273] ⌈
The E2E Manager SW-C at receiver ECU shall filter out the data elements that are not used by the SW-Cs of the ECU. The E2E Manager SW-C at receiver ECU shall forward to Conversion Manager SW-C only the data elements that are used by Conversion Manager SW-C. ⌋ ( )

12.1.4 File structure

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

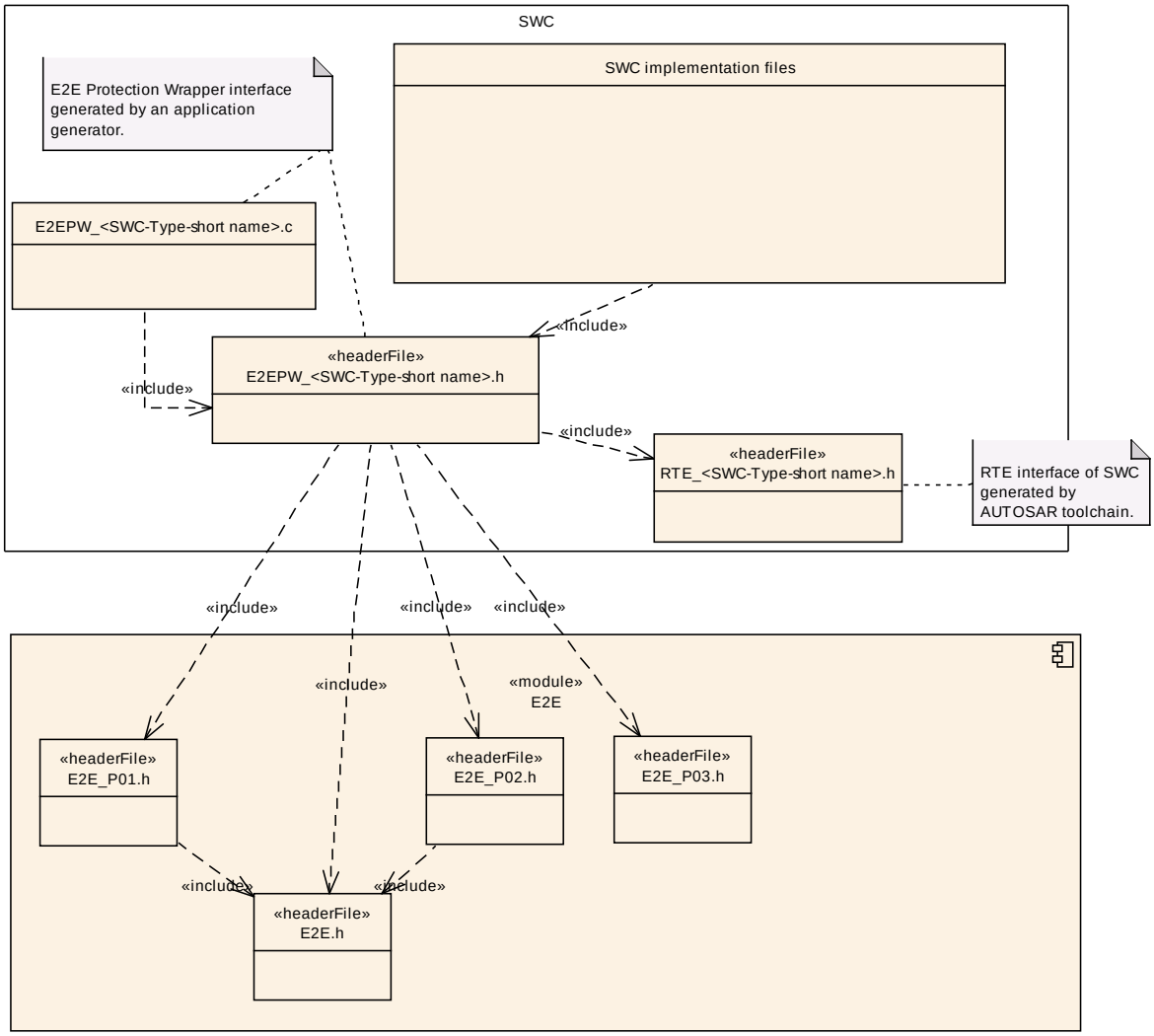


Figure 12-8: E2E File dependencies

[E2EUSE0239] ⌈
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. ⌋ ( )

[E2EUSE0240] ⌈
E2EPW_<SWC-Type-short name>.c shall include E2EPW_<SWC-Type-short name>.h. ⌋ ( )

[E2EUSE0241] ⌈
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. ⌋ ( )

[E2EUSE0242] ⌈

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

[E2EUSE0256] ⌈
The E2E Protection Wrapper shall ensure the integrity of the safety-related data elements. ⌋ ( )

[E2EUSE0257] ⌈
The implementation of the E2E Protection Wrapper (as a safety-related Software Component) shall comply with the requirements for the development of safety-related software for the automotive domain. ⌋ ( )

12.1.5 Methodology

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

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

The E2E protection wrapper can be implemented manually, or can be generated/configured from its description. All necessary information required to generate the E2E Protection Wrapper can be configured using AUTOSAR templates (system template, SW-C template, ECU configuration), with the following exception - the choice which E2E Profile is used (1, 2, 3) is not a configuration option. This is because it is assumed that in most cases the user has one wrapper generator, which generates calls to one specific E2E Profile.

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

[E2EUSE0248] ⌈
The E2E Protection Wrapper shall be generated for the complex data elements (represented by VariableDataPrototype metaclass) for which the corresponding EndToEnd* metaclasses are defined. ⌋ ( )

[E2EUSE0289] ⌈If the E2EProtection is done in the E2E Wrapper then both EndToEndProtectionISignalIPdu and EndToEndProtectionVariablePrototype shall be defined. ⌋ ( )

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

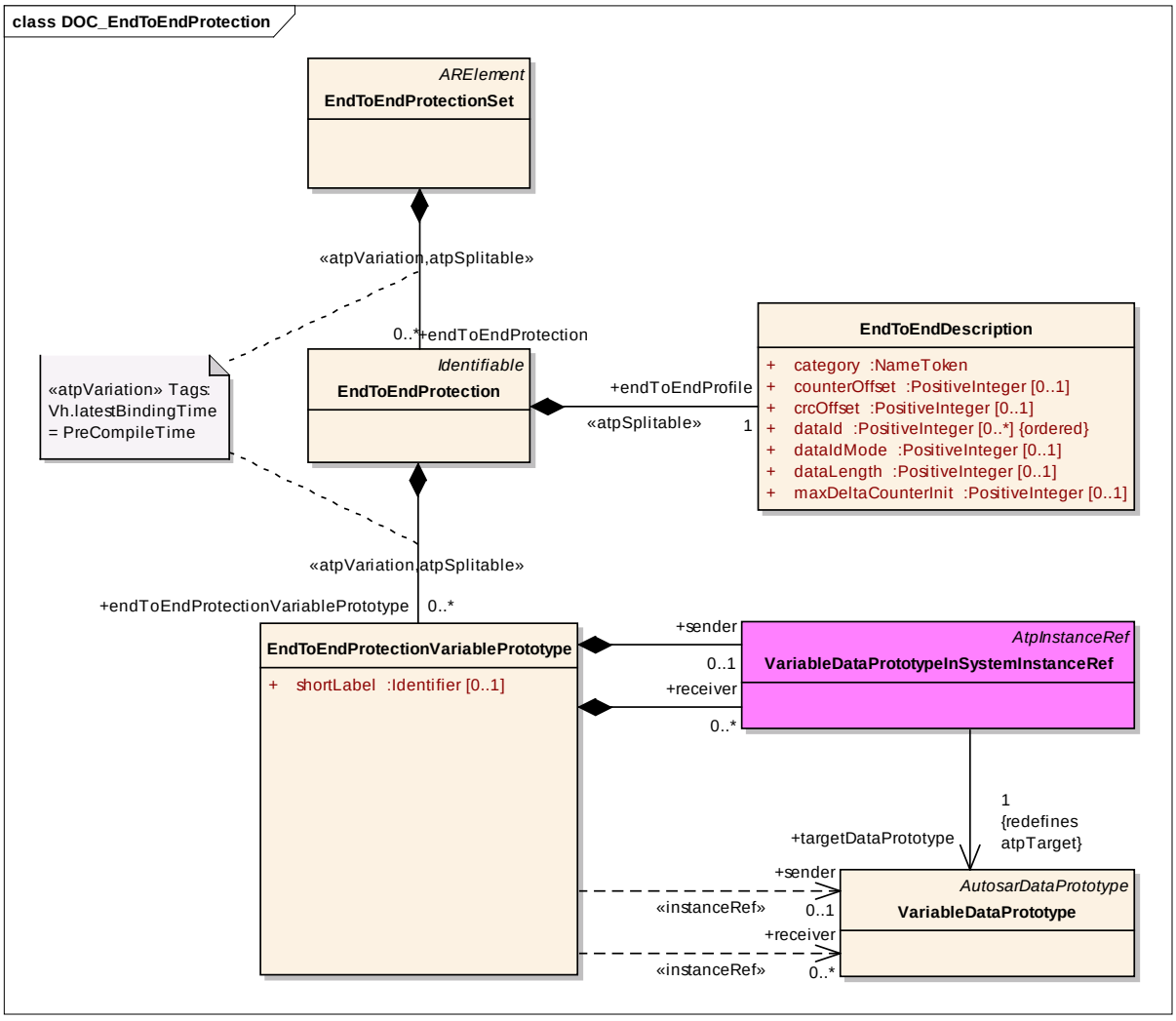


Figure 12-9: E2E Protection Wrapper configuration - SWC template

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]:

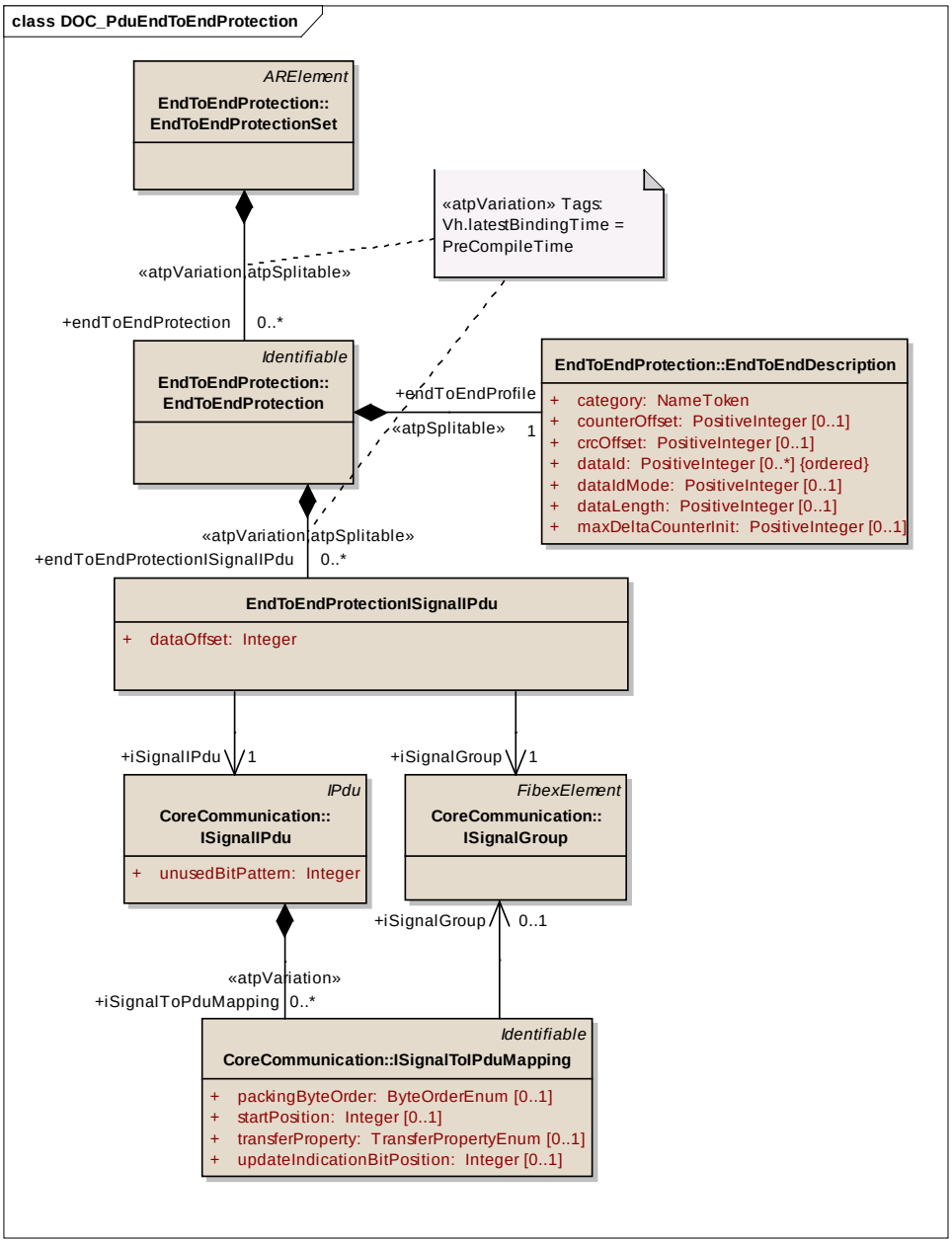


Figure 12-10: E2E Protection Wrapper configuration – System Template

The important settings are unusedBitPattern (bits that are not used in an I-PDU), startPosition (offset of the signal group in the I-PDU) and startPosition (offset of a signal in a signal group).

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

It is also necessary to configure SenderComSpec and ReceiverComSpec. ReceiverComSpec may override maxDeltaCounterInit provided by EndToEndDescription. This may be useful if different receivers (for the same sender) require a different value.

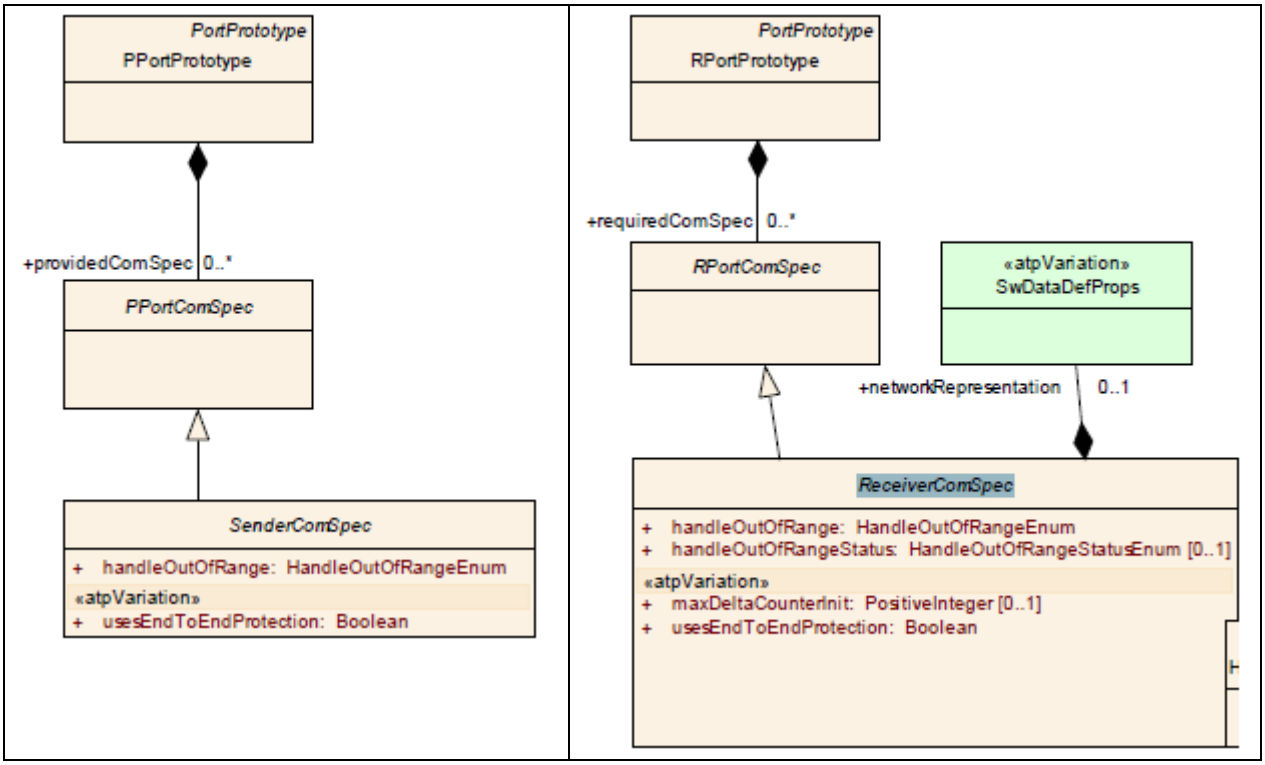


Table 12-1: SenderComSpec and ReceiverComSpec

12.1.6 Error classification

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

[E2EUSE0302]:
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 [E2E0047] (chapter 7.2.1):

Type or error or status	Relevance	Related code	Value [hex]
OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.	Production	E2EPW_STATUS_OK	0x0
Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.	Production	E2EPW_STATUS_NONEWDATA	0x1
Error: The data has been received according to communication medium,	Production	E2EPW_STATUS_WRONGCRC	0x2

but the CRC is incorrect.			
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_INITAL	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
Reserved: This value shall not be used, because in the previous versions of E2E library it was used for OK status.	Production	E2EPW_STATUS_OK_LEGACY	0x10
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	E2E_STATUS_WRONGSEQUENCE	0x40

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

[E2EUSE0303]:
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 [E2E0047] (chapter 7.2.1):

Type or error or status	Relevance	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.	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	Production	E2EPW_E_REDUNDANCY	0x5

Table 12-3: Error codes of E2E Wrapper functions (in addition to E2E Library error codes)

12.1.7 E2E Protection Wrapper routines

There are two ways how 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.

[E2EUSE0293] ⌈
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. ⌋ ( )

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.

[E2EUSE0304]:
The return values returned by the E2E Wrapper read/write functions shall be used as follows:

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

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

12.1.7.1 Single channel wrapper routines and init routines

12.1.7.1.1 E2EPW_Write_<p>_<o>

[E2EUSE0279] ⌈

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 E2E_E_INVALID - invalid value 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 E2E_E_INVALID - invalid value The byte 3 is a placeholder for future use and takes the following values E2E_E_OK - default case E2E_E_INVALID - invalid value
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.	

⌋ ( )

[E2EUSE0280] ⌈
The function E2EPW_Write_<p>_<o>() shall:

1. At first run, instantiate and initialize with initial values the data structures E2E_PXXConfigType, E2E_PXXSenderStateType, depending on the settings defined for the specific Data Element (the settings needed to generate the initialization of E2E_PXXConfigType are defined under templates System Template and Software Component Template).
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_PXXProtect().

4. Invoke Rte_Write_<p>_<o>() ⌋ ( )

12.1.7.1.2 E2EPW_WriteInit_<p>_<o>

[E2EUSE0300] ⌈

Service name:	E2EPW_WriteInit_<p>_<o>	
Syntax:	`uint8 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:	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.	

⌋ ( )

[E2EUSE0301] ⌈
The function E2EPW_WriteInit_<p>_<o> shall initialize the E2E_PXXSenderStateType_<p>_<o> with the following values:

Counter = 0⌋ ( )

12.1.7.1.3 E2EPW_Read_<p>_<o>

[E2EUSE0165] ⌈

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 E2E_E_INVALID - invalid value 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 E2E_E_INVALID - invalid value The byte 3 is the value of E2E_PXXReceiverStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC is incorrect. E2EPW_STATUS_INITAL - 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. E2E_E_INVALID - invalid value
Description:	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls optionally the corresponding function RTE_IsUpdated. Then it calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	

⌋( )

[E2EUSE0192] ⌈
The function E2EPW_Read_<p>_<o>() shall:

1. At first run, instantiate and initialize with initial values the data structures E2E_PXXConfigType, E2E_PXXReceiverStateType (the settings needed to generate the initialization of E2E_PXXConfigType are defined under templates System Template and Software Component Template).
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 optionally Rte_IsUpdated_<p>_<o>()
4. Invoke Rte_Read_<p>_<o>()
5. Invoke E2E Library function E2E_PXXCheck()⌋( )

12.1.7.1.4 E2EPW_ReadInit_<p>_<o>

[E2EUSE0296] ⌈

Service name:	E2EPW_ReadInit_<p>_<o>
Syntax:	`uint8 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:	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.	

⌋( )

[E2EUSE0297] ⌈
The function E2EPW_ReadInit_<p>_<o> shall initialize the E2E_PXXReceiverStateType_<p>_<o> with the following values:

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

12.1.7.2 Redundant wrapper routines

12.1.7.2.1 E2EPW_Write1_<p>_<o>

[E2EUSE0261] ⌈

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 E2E_E_INVALID - invalid value 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 E2E_E_INVALID - invalid value The byte 3 is a placeholder for future use and takes the following values: E2E_E_OK - default case E2E_E_INVALID - invalid value
Description:	It protects data with E2E Library function E2E_PXXProtect. it does not call the corresponding RTE_Write function.	

⌋( )

[E2EUSE0262] ⌈
The function E2EPW_Write1_<p>_<o>() shall:

1. At first run, instantiate and initialize with initial values the data structures E2E_PXXConfigType, E2E_PXXSenderStateType, depending on the settings defined for the specific Data Element (the settings needed to generate the initialization of E2E_PXXConfigType are defined under templates System Template and Software Component Template).
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_PXXProtect().⌋( )

12.1.7.2.2 E2EPW_Write2_<p>_<o>

[E2EUSE0263] ⌈

Service name:	E2EPW_Write2_<p>_<o>	

Syntax:	`uint32 E2EPW_Write2_<p>_<o>(`
	`  Rte_Instance <instance>,`
	`  -- <data>`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Write function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (inout):	<data>	Data element to be protected and sent. The parameter is inout, because this function invokes E2E_PXXProtect function, which updates the values of control fields. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (out):	None	
Return value:	uint32	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 E2E_E_INVALID - invalid value 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 E2E_E_INVALID - invalid value

		The byte 3 is a placeholder for future use and takes the following values: E2E_E_OK - default case E2E_E_INVALID - invalid value
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.	

⌋( )

[E2EUSE0264] ⌈
The function E2EPW_Write2_<p>_<o>() shall:

1. At first run, instantiate and initialize with initial values the data structures E2E_PXXConfigType, E2E_PXXSenderStateType, depending on the settings defined for the specific Data Element (the settings needed to generate the initialization of E2E_PXXConfigType are defined under templates System Template and Software Component Template).
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_PXXProtect().
4. Execute voting on control fields between Write1 and Write2.
5. Invoke Rte_Write_<p>_<o>() . ⌋( )

12.1.7.2.3 E2EPW_Read1_<p>_<o>

[E2EUSE0265] ⌈

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 E2E_E_INVALID - invalid value 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 E2E_E_INVALID - invalid value The byte 3 is the value of E2E_PXXReceiverStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC is incorrect. E2EPW_STATUS_INITAL - 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 E2E_E_INVALID - invalid value.
Description:	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls optionally the corresponding function RTE_IsUpdated. Then it calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	

⌋( )

[E2EUSE0266] ⌈
The function E2EPW_Read1_<p>_<o>() shall:

1. At first run, instantiate and initialize with initial values the data structures E2E_PXXConfigType, E2E_PXXReceiverStateType (the settings needed to generate the initialization of E2E_PXXConfigType are defined under templates System Template and Software Component Template).
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 optionally Rte_IsUpdated_<p>_<o>()
4. Invoke Rte_Read_<p>_<o>()
5. Invoke E2E Library function E2E_PXXCheck()⌋( )

12.1.7.2.4 E2EPW_Read2_<p>_<o>

[E2EUSE0267] ⌈

Service name:	E2EPW_Read2_<p>_<o>	

Syntax:	`uint32 E2EPW_Read2_<p>_<o>(`
	`  Rte_Instance <instance>,`
	`  -- <data>`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<instance>	SW-C instance. This parameter is passed to the corresponding Rte_Read function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Read function.
Parameters (inout):	None	

Parameters (out):	<data>	Parameter to pass back the received data. The pointer to the OUT. parameter <data> must remain valid until the function call returns.
Return value:	uint32	The byte 0 (lowest byte) equal to RTE_E_OK (because Rte_Read is not invoked) The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or postcondition) E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0. E2E_E_OK - Function E2EPW_Read completed successfully E2E_E_INVALID - invalid value 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 E2E_E_INVALID - invalid value The byte 3 is the value of E2E_PXXReceiverStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function. E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC is incorrect. E2EPW_STATUS_INITAL - 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 E2E_E_INVALID - invalid value.
Description:	The function re-checks the data received with corresponding function Read1 by means of execution of E2E_PXXCheck.	

⌋( )

[E2EUSE0268] ⌈
The function E2EPW_Read2_<p>_<o>() shall:

1. At first run, instantiate and initialize with initial values the data structures E2E_PXXConfigType, E2E_PXXReceiverStateType (the settings needed to generate the initialization of E2E_PXXConfigType are defined under templates System Template and Software Component Template).
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()⌋( )

12.1.8 Code Example

Important:

Because of introduction of the initialization functions in R4.0.3, all function-static variables cannot be used if initialization functions are used. All function-static variables shall be moved as module-static variables. To avoid name clashes, they shall be also suffixed. Therefore, in all code examples shown in chapter *12.1.7*, if initialization functions are used, then all static module variables shall be suffixed with either:

1. For functions E2EPW_Write_<p>_<o> and E2EPW_Read_<p>_<o>: with suffix "_<p>_<o>" (e.g. Data_<p>_<o> instead of Data)
2. For functions E2EPW_Write1_<p>_<o> and E2EPW_Read1_<p>_<o>: with suffix "1_<p>_<o>" (e.g. Data1_<p>_<o> instead of Data)
3. For functions E2EPW_Write2_<p>_<o> and E2EPW_Read2_<p>_<o>: with suffix "2_<p>_<o>" (e.g. Data2_<p>_<o> instead of Data)

To avoid making the code example too complex, the original names for variables are kept (without suffixes).

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

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

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.

The code example uses internal static variables, i.e. static variables that are private within the (wrapper) functions where they are defined. By this means, variables like Cfg, Data are all instantiated at compile time, and for each function there is a separate instance, although they share the same name.

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, therefore they are voted 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, the only data that can be voted is the results if both Read1 and Read2 functions agree on the check results (e.g. if both Read1 and Read2 report that the CRC is correct). This voting is done by comparing byte 2 of return values of Read1 and Read2 (and is executed by application (no by the wrapper).

12.1.8.1 Code Example – Sender SW-C

12.1.8.1.1 Sender – E2EPW_Write and E2EPW_Write1

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

12.1.8.1.1.1 Generation / Initialization

Generation/Initialization: RTE generates a complex data element (case A) or an opaque uit8 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 */
     uint16 speed; /* 16-bit, but 12 bits used in I-PDU*/
     uint8 accel; /* 8-bit number, 4 bits used */
} DataType;
...
DataType* AppDataEl;
```

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

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

12.1.8.1.1.2 Step S0

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

Case A (complex data type)

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

Case B (array):

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

12.1.8.1.1.3 Step S1

Step S1: Application calls E2E Protection Wrapper.

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

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

```
/* redundant – Write1 */
uint32 wapperRet1 = E2EPW_Write1_<p>_<o>(Instance, AppDataEl);
```

12.1.8.1.1.4 Step S2

Step S2: E2E Protection Wrapper (**E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()**) initializes the data structures used by E2E Library (at first run only).

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

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

12.1.8.1.1.5 Step S3

Step S3: Wrapper 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 '1'-s, as defined in unusedBitPattern of ISignalIPdu.

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

```
/* Data has the same layout as serialized signal group in I-PDU */
static uint8 Data[8];

Data[0] = 0;

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

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

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

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

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

The above example is illustrated by the figure below:

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

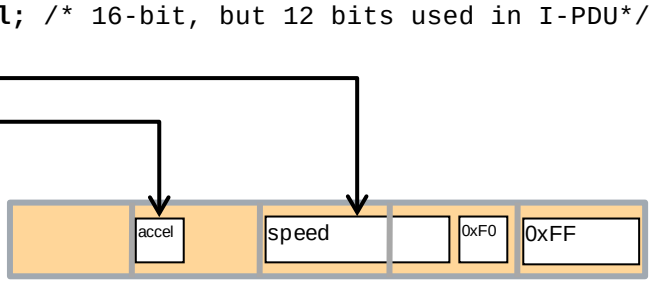


Figure 12-11: Mapping of Data elements into I-PDU

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:

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

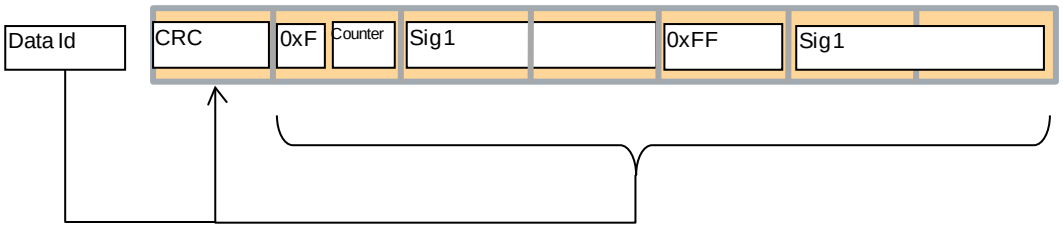
12.1.8.1.1.6 Step S4

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

```
Std_ReturnType retE2EProtect = E2E_P01Protect(Config, State, Data);
```

12.1.8.1.1.7 Step S5

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



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

Figure 12-12: Step 4

12.1.8.1.1.8 Step S6

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

Case A (complex data type):

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

This is illustrated by the Figure 12-13:

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

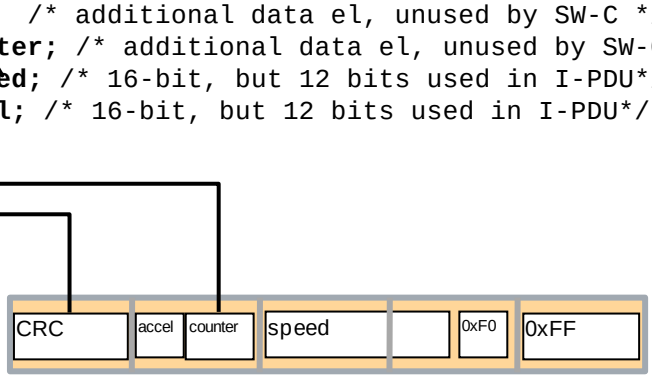


Figure 12-13: Copy back of CRC and alive from I-PDU copy to Data Element

Case B (array):

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

12.1.8.1.1.9 Step S7

Step S7: Single channel Wrapper (**E2EPW_Write_<p>_<o>**) calls RTE function to send the data element and returns the extended status to SW-C.

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

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

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

12.1.8.1.1.10 Step S8

Step S8: In case E2E Wrapper (**E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()**) has detected any runtime error in its body (e.g. wrong parameter from SW-C, violated invariant), it flags checks. Then it creates the return value and returns.

```
Std_ReturnType plausibilityChecks = E2E_E_OK;

...
/* example of possible plausibility checks */
if (AppDataEl == NULL) plausibilityChecks = E2E_E_INPUTERR_NULL;

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

12.1.8.1.1.11 Step S9

Step S9: Caller SW-C checks the return value of the wrapper and handles errors, if any. This behavior is specific to the application.

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

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

12.1.8.1.2 Sender - E2EPW_Write2

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

12.1.8.1.2.1 Step S10

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

Step S10-S19 are only for the redundant scenario. The step S10 is just the repetition of S0 on the same values. The application rewrites the data in 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)

12.1.8.1.2.2 Step S11

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

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

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

12.1.8.1.2.3 Steps S12-S15

The steps S12 to S15 (of function **E2EPW_Write2_<p>_<o>()**) are 100% identical as steps Step S2..Step S5 (of function **E2EPW_Write1_<p>_<o>()**).

12.1.8.1.2.4 Step S16 – skipped

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

12.1.8.1.2.5 Steps S17

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. 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. 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):

```
Std_ReturnType plausibilityChecks = E2E_E_OK;

/* error code – voting error between Write1 and Write2.
   The error code is different from any code returned by
   E2E_Protect() function */
#define E2EPW_E_REDUNDANCY 0xFF;

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


Std_ReturnType retRteWrite = E2E_E_OK;

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

Case B (array):

```
Std_ReturnType plausibilityChecks = E2E_E_OK;

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

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

Std_ReturnType retRteWrite = E2E_E_OK;

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

12.1.8.1.2.6 Step S18

Step S18: In case E2E Wrapper (**E2EPW_Write_<p>_<o>, E2EPW_Write1_<p>_<o>()**) has detected any runtime error in its body (e.g. wrong parameter from SW-C, violated invariant), it flags checks. Then it creates the return value and returns.

```
...
/* example of possible plausibility checks. */
if (AppDataEl == NULL) plausibilityChecks = E2E_E_INPUTERR_NULL;

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

12.1.8.1.2.7 Step S19

Step S9: Caller SW-C checks the return value (of function **E2EPW_Write2_<p>_<o>()**) and handles errors, if any. It also compares the return values of **E2EPW_Write2_<p>_<o>()** against return value of **E2EPW_Write1_<p>_<o>()** .

```
if(wrapperRet2 != 0) swc_error_handler(ret2);
```

12.1.8.2 Code Example – Receiver SW-C

12.1.8.2.1 Receiver - E2EPW_Read and E2EPW_Read1

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

12.1.8.2.1.1 Generation / Initialization

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

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

```
typedef struct {
     uint8 crc;     /* additional data el, unused by SW-C */
     uint8 counter; /* additional data el, unused by SW-C */
     uint16 speed;  /* 16-bit, but 12 bits used in I-PDU*/
     uint8 accel;   /* 16-bit, but 12 bits used in I-PDU*/
     } DataType;
     ...
     DataType* AppDataEl;
```

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

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

12.1.8.2.1.2 Step R1

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

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

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

12.1.8.2.1.3 Step R2.0

Step R2.0: E2E Protection Wrapper (**E2EPW_Read_<p>_<o>, E2EPW_Read1_<p>_<o>()**) initializes the data structures used by E2E Library (at first run only)

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


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

12.1.8.2.1.4 Step R2.1

Step R2.1: Wrapper (**E2EPW_Read_<p>_<o>, E2EPW_Read1_<p>_<o>()**) calls RTE functions **Rte_IsUpdated** (optionally) and then **Rte_Read** to receive the data element.

```
/* set it to:
   TRUE - if Rte_IsUpdated is available in your RTE (AUTOSAR >=4.0)
   FALSE - for AUTOAR <4.0 or if SW-C invokes the wrapper after
   having been notified that new data is available */
#define E2E_USING_RTE_ISUPDATED TRUE


/* Check first if RTE has new data */
#if (E2E_USING_RTE_ISUPDATED)
State->NewDataAvailable = Rte_IsUpdated_<p>_<o>(Instance, AppDataEl);
#else
State->NewDataAvailable = TRUE;
#endif

Std_ReturnType retRteRead = RTE_E_OK;

/* If new data available, then read it */
if (State->NewDataAvailable) {
      retRteRead = Rte_Read_<p>_<o>(Instance, AppDataEl);
}
```

12.1.8.2.1.5 Step R3

Step R3: the E2E Protection Wrapper serializes the data to the layout identical with the one of the corresponding I-PDU. The E2E Protection wrapper needs to do the serialization (I-PDU from the received data), so that E2E Library can compute and check the CRC.

Case A (complex data type):

```
/* For storing the same layout as the one of I-PDU */
static uint8 Data[8];

Data[0] = 0;

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

/* in speed, only 8+4 bits are used. */
  low byte of speed goes to Data[2]./
```

```
Data[2] = (AppDataEl->speed & 0x00FF);

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

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

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

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

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

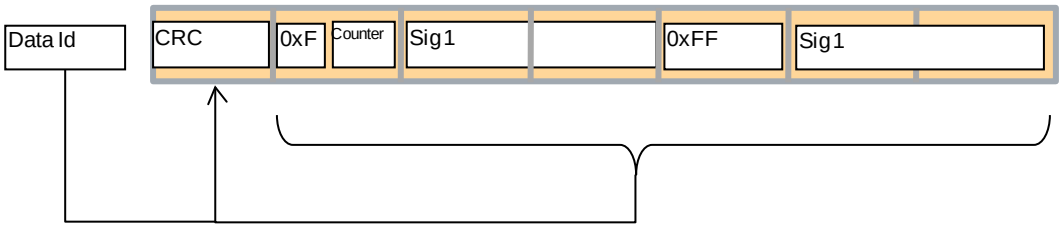
12.1.8.2.1.6 Step R4

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

```
Std_ReturnType retE2ECheck = E2E_P01Check(Config, State, Data);
```

12.1.8.2.1.7 Step R5

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



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

12.1.8.2.1.8 Step R6 – skipped

No control fields need to be copied to **AppDataEl**, as they are only verified.

12.1.8.2.1.9 Step R7

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

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

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

...
Std_ReturnType plausibilityChecks = E2E_E_OK;

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

if( (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

```
Std_ReturnType plausibilityChecks = 0;
```

12.1.8.2.1.10 Step R8

Step R8: In case E2E Wrapper detects any runtime error in its body (e.g. wrong parameter from SW-C, program flow error), it flags checks. Then the wrapper returns to the application.

```
Std_ReturnType plausibilityChecks = 0;
...
/* example check */
if (AppDataEl == NULL) plausibilityChecks = E2E_E_INPUTERR_NULL;

/* note that the last byte is the  */
return ( (retRteRead) | (retE2ECheck<<8) |
     (plausibilityChecks<<16) | (uint8)(State->Status) );
```

12.1.8.2.1.11 Step R9

Step R9: Caller SW-C checks the return value and handles errors, if any. This behavior is specific to the application. Then it copies the data from **`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 */
/* simple example: on byte 3, only E2E_PXXSTATUS_OK is accepted*/
if(wrapperRet != 0) swc_error_handler(ret);
targetSpeed = AppDataEl->speed;
targetAccel = AppDataEl->accel;
```

```
/* redundant */
if(wrapperRet1 != 0) swc_error_handler(ret1);
targetSpeed1 = AppDataEl->speed;
targetAccel1 = AppDataEl->accel;
```

Case B (array):

```
/* single channel */
if(wrapperRet != 0) swc_error_handler(ret);
targetSpeed = (AppDataEl[2]) | (AppDataEl[3]<<8 & 0x0F);
targetAccel = AppDataEl[1] >> 4;
```

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

12.1.8.2.2 Receiver - E2EPW_Read2

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

12.1.8.2.2.1 Step R10 – skipped

Value unused to numbering consistency.

12.1.8.2.2.2 Step R11

Step R11: Application calls the wrapper again.

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

12.1.8.2.2.3 Step R12.0

The step R12.0 (of function **`E2EPW_Read2_<p>_<o>()`**) are 100% identical to step Step R2.0 (of function **`E2EPW_Write2_<p>_<o>()`**).

12.1.8.2.2.4 Step R12.1 – skipped

Contrary to Step R2.1, RTE is not read. Both read steps use the same data from RTE, which are read in step Step R2.1.

12.1.8.2.2.5 Steps R13-R15

The steps R13 to R15 (of function **`E2EPW_Read2_<p>_<o>()`**) are 100% identical as steps Step R3 .. Step R5 (of function **`E2EPW_Read1_<p>_<o>()`**).

12.1.8.2.2.6 Step R16 – skipped

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

12.1.8.2.2.7 Step R17

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

12.1.8.2.2.8 Step R18

The step R18 (of function **`E2EPW_Read2_<p>_<o>()`**) are 100% identical to step **Step R8** (of function **`E2EPW_Read1_<p>_<o>()`**).

12.1.8.2.2.9 Step R19

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

Case A (complex data type):

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


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

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

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

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

Case B (array):

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


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

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


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

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

12.2 COM E2E Callouts

In this approach, the E2E communication protection protects the data exchange between COM modules. The protection is done at the level of COM's I-PDUs, which are protected and checked by E2E Library.

This solution works with all communication models, multiplicities offered by RTE for inter-ECU communication.

It is possible that the E2E Library is invoked by COM, through COM E2E callouts, to protect/check the I-PDUs. The callout invokes the E2E Library with parameters appropriate for 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 to be protected/checked, there is a separate callout function. Each callout function “knows” how each I-PDU needs to be protected/checked. This means that the callout invokes the E2E Library functions with settings and state parameters that are appropriate for the given I-PDU. The E2E Library does now “know” I-PDUs and their settings – E2E Library has no configuration.

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 unimpacted, 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 IPDU in this cycle.

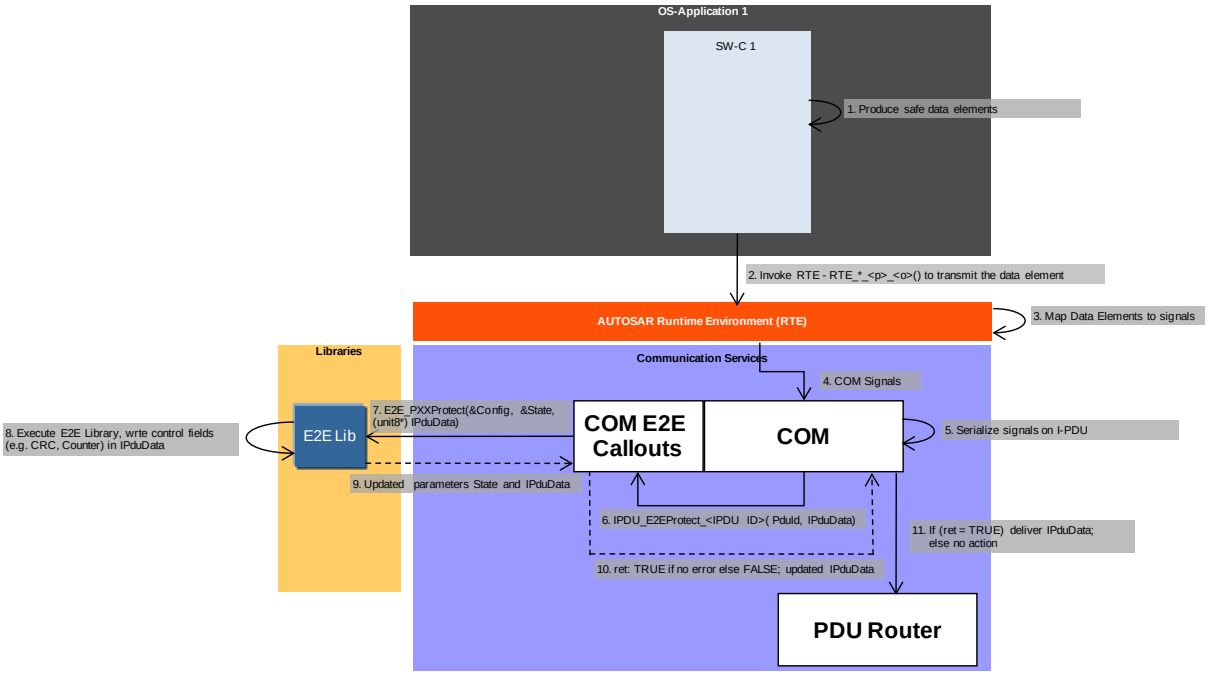


Figure 12-14: Callout – overall flow – P-port

The diagram below summarizes the COM E2E Callout solution. The very important step is that the E2E Library overwrites CRC byte in the I-PDU by the check status bits (E2E_PXXReceiverStateType). Then, this overwritten CRC byte is converted by

COM to signals and then by RTE to data elements. As a result, the SW-C receives in the CRC data element the E2E check bits, and not the CRC value.

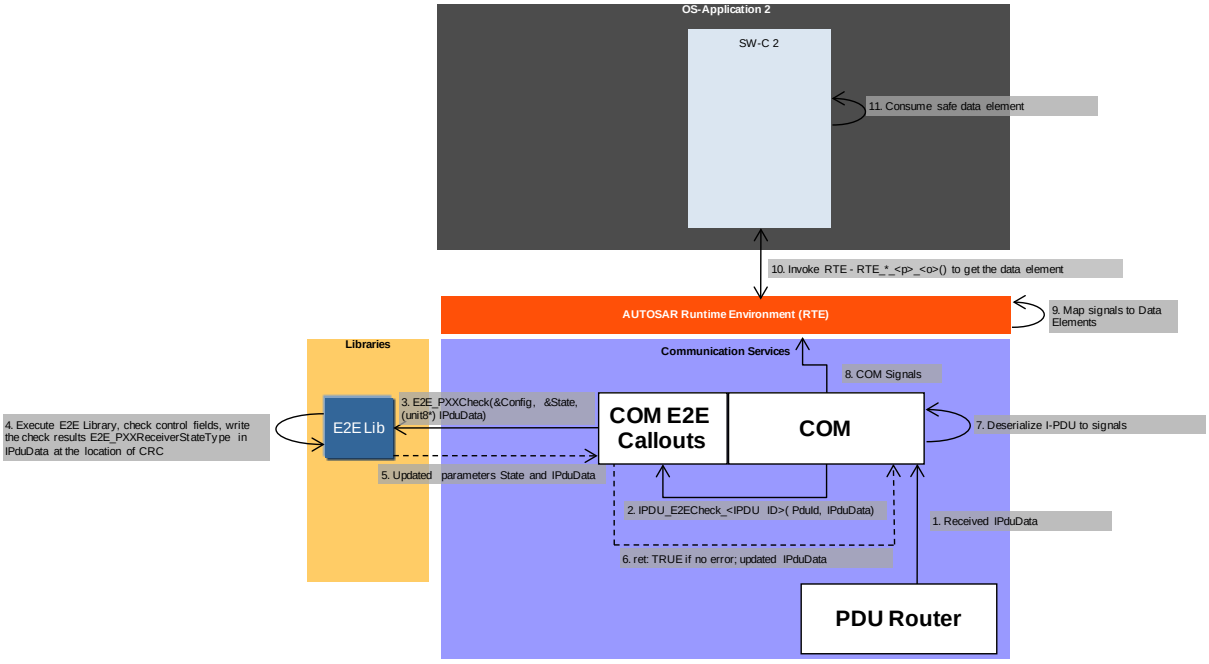


Figure 12-15: Callout – overall flow – R-port

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 specifically one I-PDU and simply invokes the E2E Library 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.

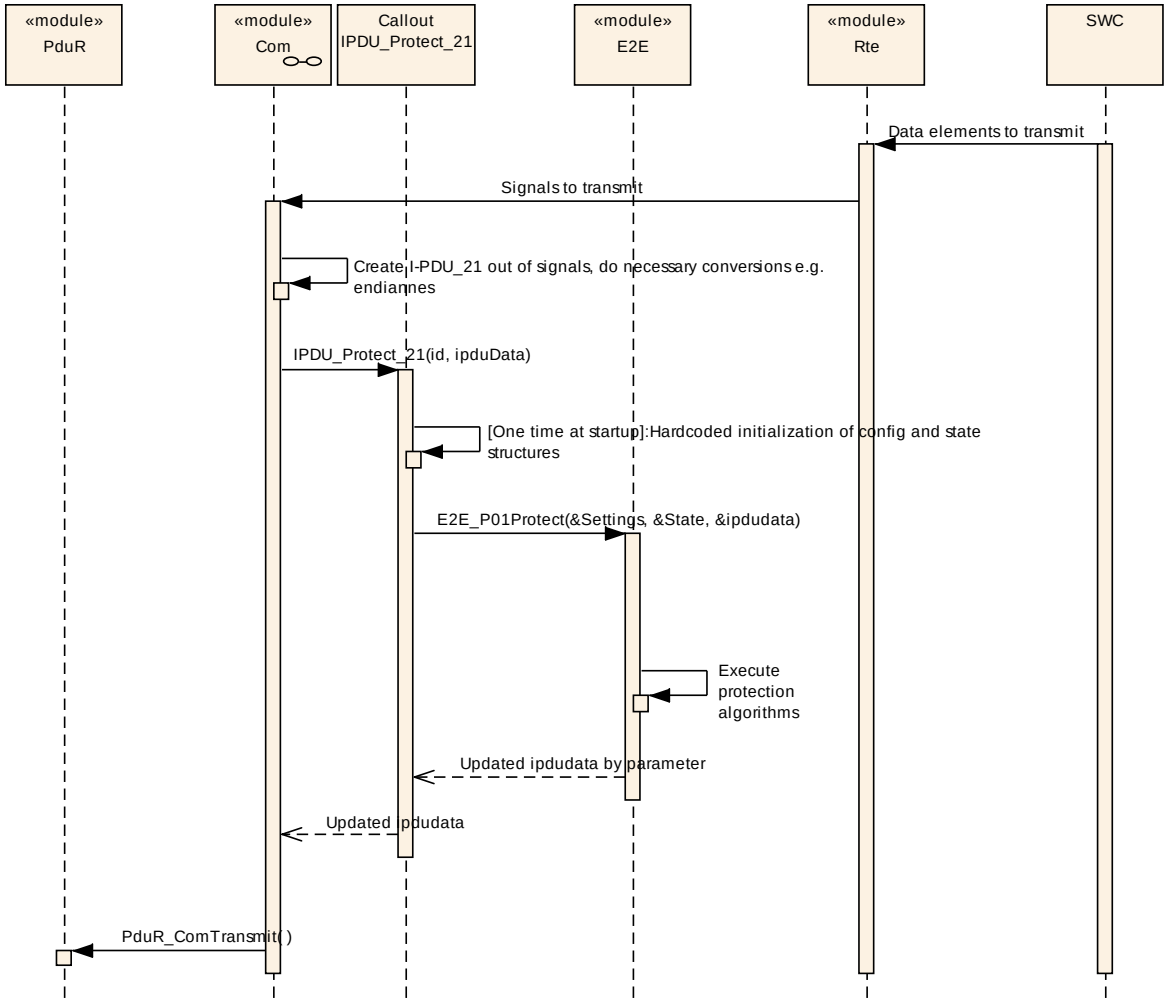


Figure 12-16: Callout – sequence – sending

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)

[E2EUSE0250] ⌈
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 PDU with handle 21 shall have the name `IPDU_E2EProtect_21().`⌋ ()

Reception

On the receiver COM side, when the I-PDU is available at PDU Router, PDU Router invokes COM's function COM_RxIndication(). COM then calls the generated I-PDU

callout (if configured for the given I-PDU). The callout, generated specificallly for that I-PDU, calls the E2E Library with specific parameters. The E2E Library executes the check paramters 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 an be analyzed, if needed, by receiver SW-C.

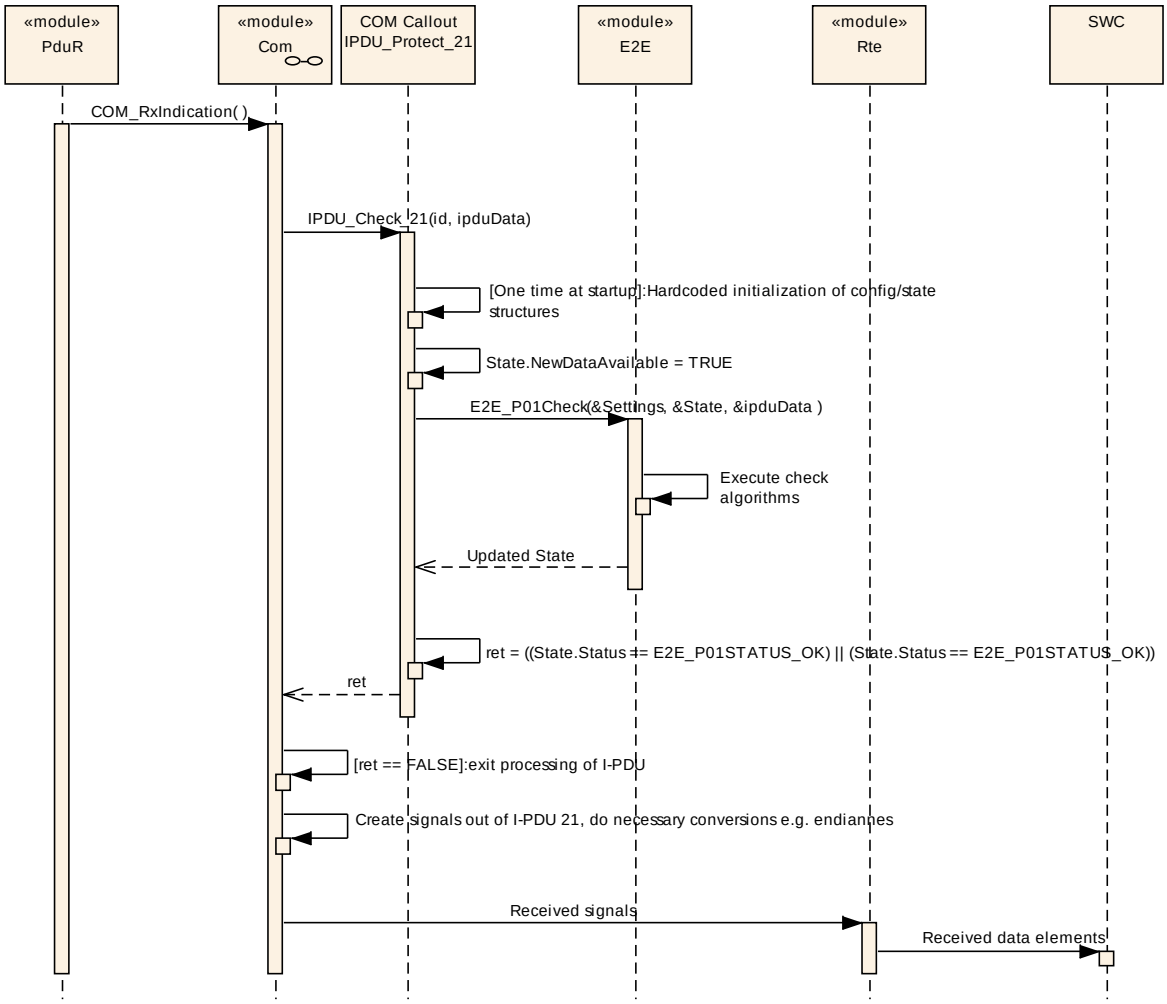


Figure 12-17: Callout - sequence - reception

[E2EUSE0251] ⌈
The reception callout for usage with E2E shall be the following: `IPDU_E2Echeck_<IPDU ID>().`

For example, the callout to protect the PDU with handle 21 shall have the name `IPDU_E2ECheck_21().`⌋`()`

12.2.2 Methodology

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

The information how each I-PDU needs to be protected (e.g. which E2E Profile, which offset) are 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 I-PDU 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 information needed to generate the COM callouts automatically is available. For each I-PDU to be protected/checked, there is to be generated a separate callout routine, that invokes E2E Library. Each callout simply invokes the E2E Library routines each with different settings.

[E2EUSE0270] ⌈
The COM E2E callout shall be generated for the I-PDU for which the corresponding EndToEnd* metaclasses are defined. ⌋ ( )

[E2EUSE0290] ⌈If the E2EProtection is done via COM Callouts then the EndToEndProtectionISignalIPdu shall be defined. ⌋ ( )

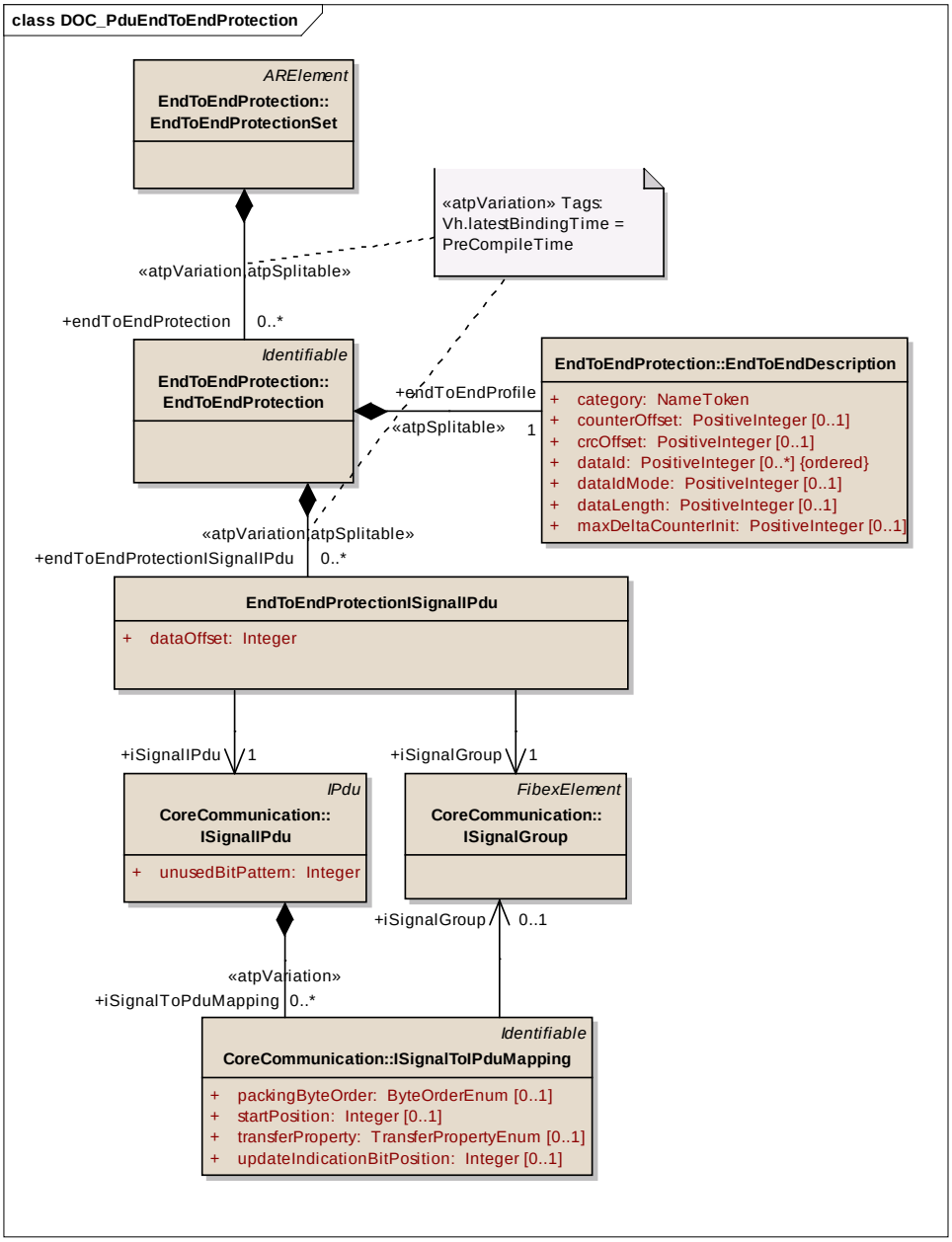


Figure 12-18: COM Callouts Configuration (hardcopy)

Information	Details
Definition of E2E Protection settings (separate settings for each Signal Group)	Class EndToEndDescription
Definition of I-PDUs unused bits	Attribute unusedBitPattern
Association of Signal Groups to I-PDUs including bit offset of the Signal Group	Attribute startPosition
Association which protection settings shall be used for a given Signal Group	EndToEndProtectionSignalIPdu
Definition of numeric handles / identifiers of I-PDUs. The handles are used to identify the I-PDU when invoking the	I-PDU Handle ID (not shown on the diagram)

callouts	
Definition of callouts to be invoked to protect/check an I-PDU. For each I-PDU, there is a separate callout.	Class ComIPDU, with attribute ComIPduCallout (not shown on the diagram).

Table 12-4: Settings and templates

12.2.3 Code Example

Transmitter

```
FUNC(boolean, COM_APPL_CODE) IPDU_E2EProtect_21 (PduIdType id, P2VAR
(uint8, AUTOMATIC, COM_VAR_NOINIT) ipduData){

     /* At first run, instantiate the structures and set the init
        values*/
     static E2E_P01ConfigType Cfg_Write_21 =
                          { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
     static E2E_P01SenderStateType Sta_Write_21 = {0};


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

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

}
```

Receiver

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

     /* At first run, instantiate the structures and set the init
values*/
     static E2E_P01ConfigType Cfg_Read_21  =
                       { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
     static E2E_P01ReceiverStateType Sta_Read_21 =
                       {0, 0, TRUE, FALSE, E2E_P01STATUS_NONEWDATA };

     /* If callout is invoked, this means that new data is available
     At COM */
     Sta_Read_21.NewDataAvailable = TRUE;

     Std_ReturnType ret = E2E_P01Check(Cfg_Read_21, Sta_Read_21, ipduData);


     /* return TRUE if no error, possibly only some messages lost
```

```
      Within counter tolerance */
      if(   ret == E2E_OK &&
            (Sta_Read_21.Status ==  E2E_P01STATUS_OK ||
             Sta_Read_21.Status ==  E2E_P02STATUS_OKSOMELOST)   ) {

            return TRUE;
      }
      else {

            return FALSE;
      }
}
```

12.3 Provision of the Protection Wrapper Interface on a ECU with COM Callout solution

In case an ECU can provide a safe hardware, COM Layer and RTE, it is possible to integrate SWCs which require the E2E Protection Wrapper interfaces by using a direct mapping of E2E Wrapper interfaces to RTE interfaces and perform the E2E protection according to the "COM Callout" approach. By this approach compatibility between the two solutions "E2E Protection Wrapper" and "COM Callout" is achieved. This implies that the CRC and Ctr fields are not yet filled on RTE level in Tx direction. For Rx direction the CRC and Ctr on RTE level are already evaluated by COM and filled with status information and thus do not contain the PDU checksum and counter anymore.

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	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
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		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		
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**																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
101	218	39	57	175	229	13	160	15	177	49	62	40	240	145	82	209	X	X	X	X	X	X		X	X				X	X	X	X		X			X	X	X		X	X										X		X		X	X
102	8	140	86	199	46	103	122	242	30	91	84	159	161	41	193	165	X		X		X	X	X	X	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	31	56	219	247	199	232	226	192	142	242	72	64	148	62	159	166	X	X		X		X	X	X	X	X	X	X	X						X		X					X	X				X					X	X		X	X	X
104	184	223	234	215	125	46	88	102	129	130	31	91	222	218	213	244		X				X			X	X		X		X	X		X	X			X	X		X	X	X		X			X	X	X	X					X		
105	135	109	197	76	64	158	74	163	93	222	50	161	57	126	172	44	X	X	X			X	X		X	X	X		X				X		X			X	X	X	X		X	X	X							X	X		X	X	X
106	62	98	227	24	69	22	198	231	68	239	12	54	169	153	186	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	
107	44	152	19	219	86	67	211	232	230	103	39	142	116	224	30	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
108	105	70	233	191	195	176	232	250	132	59	200	138	213	64	183	60	X					X	X				X	X	X	X	X					X		X	X	X				X	X	X	X	X		X	X		X		X	X	
109	124	58	154	220	37	212	125	188	24	248	77	171	146	130	50	204		X	X	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	215	102	91	244	139	168	39	200	5	187	247	167	73	175	78	43	X	X	X					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	92	9	113	47	35	115	97	37	21	72	147	101	1	75	175	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				X
112	85	29	163	68	28	44	157	31	33	8	196	152	15	1	73	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	X	X	X		X
113	249	55	32	87	111	141	9	57	242	7	211	114	68	47	244	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	X	
114	17	195	94	59	1	183	159	146	190	208	123	38	245	194	211	66	X	X	X			X			X	X		X	X	X	X		X	X			X	X			X		X	X				X			X	X	X			X	
115	38	18	149	74	133	178	77	126	48	227	11	27	47	52	29	6	X		X	X		X		X	X				X		X	X			X		X						X	X		X	X		X			X		X			
116	69	239	186	249	107	170	204	55	140	116	6	32	85	92	215	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		X		X
117	233	176	132	138	183	160	96	120	37	38	80	145	162	172	169	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	X		X	X	X
118	56	247	232	192	242	64	62	166	96	165	251	222	221	98	194	28			X	X	X	X				X	X		X	X	X		X	X			X					X		X	X	X	X			X	X				X	X	
119	168	167	217	80	206	144	40	63	51	3	222	225	30	107	58	241		X		X	X		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	7	81	200	82	230	43	251	100	80	179	156	237	199	79	97	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
121	113	115	21	101	175	69	23	50	214	15	112	239	211	11	40	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
122	170	32	199	141	102	242	113	114	159	244	189	165	44	115	168	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	X
123	108	20	60	173	147	89	28	77	216	51	93	137	123	8	112	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	X			X		X
124	94	183	190	38	211	169	20	18	251	224	3	149	92	173	117	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	X	X	X
125	194	136	173	95	144	77	186	205	137	225	161	52	179	170	26	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	X
126	217	144	51	225	58	26	223	198	181	220	238	4	250	215	212	153		X		X		X		X	X			X	X	X		X	X		X		X	X	X		X	X			X	X						X	X	X	X		X
127	216	12	34	243	126	207	14	94	246	6	235	201	220	30	203	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	X	X
128	171	110	71	25	88	122	78	113	118	218	59	41	126	206	87	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	X			X		X
129	211	224	117	133	131	85	51	227	76	156	145	29	35	26	166	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	X
130	141	114	165	177	200	90	101	184	136	43	85	119	219	50	80	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
131	20	173	89	77	51	137	8	52	12	26	150	17	97	140	181	10	X			X		X				X		X		X	X	X			X	X		X	X	X		X	X		X	X	X	X				X		X	X	X	
132	206	3	58	23	160	220	93	11	212	145	95	188	210	150	248	245		X	X	X		X			X	X	X	X	X					X	X	X			X	X	X		X	X		X			X	X	X			X			X
133	129	193	127	5	162	210	114	217	146	174	142	147	105	177	45	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	X	X
134	201	190	72	169	189	251	60	149	15	117	58	79	25	89	109	178	X			X		X	X	X		X				X			X	X	X		X	X		X	X		X			X	X		X	X	X		X	X		X	
135	147	51	112	26	154	181	234	4	126	212	182	157	138	46	24	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
136	18	74	178	126	227	27	52	6	185	22	245	203	37	10	68	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
137	161	33	151	19	104	2	16	67	187	14	87	230	223	189	70	142		X	X	X		X			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	120	216	202	12	74	34	104	243	196	126	213	207	58	14	27	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
139	232	64	96	222	194	172	227	164	61	136	209	238	81	22	173	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
140	68	31	152	56	140	219	208	247	67	199	35	232	99	226	103	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			X	X	X	
141	214	186	8	170	223	140	110	32	86	215	203	199	76	25	46	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	166	28	164	8	119	73	231	140	104	234	185	86	197	71	205	199			X		X					X	X					X	X		X	X	X		X		X	X	X		X		X	X			X		X	X	X	X	X
143	156	214	28	186	184	8	171	170	73	223	27	140	109	110	234	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	X	X			X	X	
144	72	251	15	79	109	209	216	48	107	76	212	99	41	12	158	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		
145	152	219	67	232	103	142	224	64	179	159	175	96	111	133	108	222	X	X	X	X	X		X				X	X				X	X	X	X	X	X				X			X		X	X			X		X	X	X	X		
146	241	213	153	235	245	162	7	78	42	53	103	174	77	81	92	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
147	145	202	236	34	178	196	105	207	88	27	153	246	154	70	185	201	X	X			X	X	X		X	X		X		X	X	X		X						X	X		X			X				X		X		X	X		
148	234	46	129	91	213	193	87	168	127	235	219	5	238	57	162	167		X		X	X	X	X		X	X	X		X				X	X	X		X		X		X	X			X				X			X	X	X		X	
149	154	212	24	171	50	231	213	110	54	83	17	71	18	235	249	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	X	X	
150	10	127	59	210	42	146	165	147	38	45	179	66	134	90	226	51		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
151	2	230	123	179	191	97	109	128	3	250	177	75	91	76	138	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
152	52	10	195	127	65	59	242	210	183	42	230	146	180	165	208	147					X		X		X			X	X	X	X		X	X	X					X	X				X	X		X	X			X		X	X	X	
153	43	237	63	125	75	241	99	130	198	228	8	213	194	148	11	139		X	X			X		X			X	X	X		X				X	X	X			X		X		X	X	X			X		X	X		X		X	X
154	86	103	30	159	193	108	131	194	250	5	49	20	151	156	210	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	X	
155	14	30	191	108	127	250	192	20	138	210	82	60	139	166	146	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
156	109	76	158	163	222	161	126	44	150	164	83	33	13	6	238	152	X		X		X		X		X	X		X		X		X	X	X			X	X		X	X	X	X							X	X			X		X	
157	79	48	99	185	163	148	243	36	111	44	110	221	69	94	33	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
158	185	36	221	84	152	229	183	135	81	219	113	49	249	38	67	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	
159	26	4	157	65	171	1	91	42	143	110	70	208	216	168	71	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			X
160	193	5	210	217	174	147	177	144	66	124	96	51	70	184	106	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
161	5	217	147	144	124	51	184	225	112	58	172	26	191	223	154	198			X		X	X	X	X	X		X		X	X	X		X		X	X	X	X	X						X	X	X			X		X			X		X
162	82	100	237	93	128	125	48	150	241	61	186	130	165	185	228	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	X	X
163	95	205	52	129	198	10	32	193	195	153	19	127	61	141	65	5			X	X	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	102	244	168	200	187	167	175	43	217	123	192	80	86	15	206	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
165	231	71	240	122	55	118	174	41	247	87	183	131	27	124	141	21	X		X		X						X	X	X	X	X	X			X			X	X			X	X				X	X	X	X	X	X		X			X
166	96	172	61	238	173	182	68	104	180	77	116	105	43	31	137	14		X	X			X	X			X		X				X		X	X				X		X	X			X	X		X	X		X		X	X	X		X
167	90	119	95	234	63	205	249	46	52	241	44	129	96	55	198	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	X	X
168	127	210	146	147	45	66	90	51	18	106	128	112	233	119	62	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			X	X	
169	176	138	160	120	38	145	172	216	248	18	63	202	174	238	149	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	X	X	X		X
170	63	241	198	213	11	153	111	235	65	245	86	162	173	7	204	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
171	41	21	156	69	13	214	220	239	28	40	74	186	189	188	184	249			X	X	X	X	X	X	X		X		X		X		X	X		X	X		X	X	X	X		X					X		X			X			X
172	162	174	45	124	9	106	43	58	62	47	20	154	195	237	115	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				X	
173	142	96	128	172	20	61	29	238	228	173	107	182	200	68	89	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
174	205	129	10	193	153	127	141	5	59	162	67	210	182	114	42	217	X	X	X	X	X		X		X	X	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	75	228	11	180	216	245	33	134	204	12	46	53	144	19	34	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	X	X	X	
176	178	27	185	203	68	36	195	16	221	31	92	84	50	59	152	189	X		X	X		X	X	X		X	X		X	X		X				X	X	X	X		X		X			X	X						X				X
177	225	198	4	153	188	65	102	162	1	204	14	42	89	244	110	174	X		X	X	X	X	X							X		X		X	X	X			X		X	X	X		X		X	X	X						X	X	X
178	32	141	242	114	244	165	115	177	194	200	117	90	152	101	167	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		X			X
179	51	26	181	4	212	157	46	65	6	171	105	1	120	91	231	42		X	X		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	54	240	56	118	32	247	45	131	232	141	190	192	185	106	242	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		
181	223	215	46	102	130	91	218	244	193	139	56	168	164	39	235	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
182	101	50	239	83	209	249	245	88	170	99	157	55	133	53	116	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		
183	58	220	212	188	248	171	130	204	231	236	52	110	66	139	83	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
184	179	128	75	61	60	228	163	182	11	89	223	180	167	44	216	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	
185	110	25	122	113	218	41	206	115	131	39	146	21	6	3	57	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			X
186	71	122	118	41	87	131	124	21	192	57	38	156	203	58	114	69			X	X		X	X	X		X	X			X		X			X	X	X	X			X					X		X			X		X	X	X	X	
187	153	162	42	174	92	45	200	124	226	9	108	106	17	43	113	58			X	X	X			X	X	X	X			X	X	X	X	X				X				X					X	X	X		X	X	X			X	
188	119	234	205	46	241	129	55	91	10	213	152	193	172	87	153	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		X	X	X
189	91	168	5	167	78	217	13	80	147	206	64	144	14	40	124	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	X
190	190	169	251	149	117	79	89	178	209	85	220	48	113	137	76	27	X	X	X	X				X			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	187	123	206	97	176	3	197	75	58	138	90	23	193	158	160	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	
192	165	90	136	119	80	95	239	234	77	63	163	205	142	249	225	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
193	60	89	216	137	112	12	73	17	34	181	130	243	3	86	126	195	X						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	98	24	22	231	239	54	153	71	31	249	243	240	149	162	170	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
195	140	199	103	242	91	159	41	165	108	168	135	194	33	21	5	90					X	X	X	X	X	X	X	X				X	X	X	X	X	X	X			X	X		X	X			X	X							X	X
196	139	187	78	123	233	206	49	97	124	176	165	3	129	197	132	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	
197	130	139	235	187	134	78	229	123	174	233	242	206	205	49	121	97	X		X	X		X		X		X		X				X	X					X		X			X		X			X	X	X	X				X	X	
198	230	179	97	128	250	75	76	61	23	60	184	228	168	163	120	182			X	X	X	X		X		X		X				X	X				X	X	X	X	X		X				X	X		X	X	X	X			X	X
199	16	189	135	117	247	109	66	85	197	192	37	76	218	112	64	29	X				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	84	135	49	109	232	197	18	76	100	64	101	158	87	74	96	163	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
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
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	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**

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
351	109	105	68	172	122	115	6	156	48	23	18	46	139	239	167	207
352	187	251	42	144	101	128	190	47	121	28	73	234	71	14	36	158
353	233	4	108	241	144	203	59	135	12	47	82	85	33	18	234	218
354	174	54	61	246	216	101	80	151	160	142	233	28	170	109	86	81
355	213	63	189	247	85	37	112	94	166	170	39	71	68	98	70	97
356	18	181	132	27	25	129	245	232	165	143	164	124	195	173	49	248
357	97	133	186	108	42	222	13	203	131	128	105	12	102	194	121	85
358	76	217	90	199	196	154	203	87	228	215	72	92	86	85	93	32
359	193	96	119	175	117	159	216	179	27	64	30	224	116	142	39	103
360	246	151	28	81	58	118	105	72	243	83	241	230	202	172	138	140
361	212	114	134	225	168	31	183	117	77	82	99	182	22	136	172	64
362	16	197	94	165	50	97	205	233	229	187	130	133	156	22	202	4
363	141	9	57	242	7	211	114	68	47	244	13	171	165	225	81	115
364	125	196	192	215	22	93	234	116	236	223	100	107	184	40	13	155
365	107	146	226	112	75	95	30	98	110	132	162	157	225	219	197	180
366	25	143	49	141	150	29	103	9	251	102	180	57	16	212	246	242
367	90	154	228	92	93	6	5	55	117	107	188	239	21	70	195	250
368	4	241	203	135	47	85	18	218	5	158	123	170	201	181	40	50
369	118	230	163	188	137	79	46	235	210	104	170	177	121	150	96	65
370	244	2	72	167	131	140	123	88	188	15	238	191	144	44	183	166
371	134	31	77	182	172	25	41	39	3	156	66	143	13	162	46	220
372	29	57	144	211	151	47	134	171	234	81	11	158	97	31	118	48
373	227	1	32	53	99	249	50	30	250	210	166	120	100	187	66	219
374	192	93	236	107	13	195	1	146	123	194	104	226	212	53	198	112
375	40	38	53	62	79	30	49	126	120	177	71	219	15	29	106	160
376	163	79	210	177	96	106	54	245	74	175	237	176	87	246	159	173
377	250	164	206	84	98	214	169	192	240	180	204	19	175	121	20	93
378	224	152	248	52	80	190	138	134	8	109	215	14	98	137	242	31
379	13	194	198	75	43	16	210	132	247	238	239	197	168	106	218	129
380	211	171	158	48	72	127	182	174	38	140	16	51	108	39	188	54
381	1	53	249	30	210	120	187	219	164	106	91	125	183	251	74	231
382	137	104	96	41	6	175	166	162	159	239	62	179	148	91	64	184
383	160	36	161	243	21	148	171	163	116	185	12	99	76	48	209	79
384	19	236	82	195	11	123	32	226	156	198	96	44	52	249	63	95
385	99	210	66	106	119	74	61	176	98	159	56	205	92	101	27	78
386	146	112	95	98	132	157	219	180	69	129	184	20	117	231	165	113
387	71	229	35	186	17	200	153	222	83	169	242	131	54	11	217	12
388	55	250	89	164	112	206	17	84	130	98	86	214	96	169	157	192
389	203	85	5	170	40	70	132	60	26	38	207	237	2	129	53	202
390	165	233	133	4	251	108	223	241	222	144	168	203	221	59	128	135
391	3	49	187	29	54	251	152	57	42	246	20	144	247	52	101	211
392	218	50	60	187	51	202	232	251	17	61	69	42	189	248	126	144
393	238	3	50	49	174	187	224	29	202	54	157	251	63	152	61	57
394	82	123	156	44	63	207	250	110	150	247	159	221	14	164	37	69
395	133	108	222	203	128	12	194	85	15	234	172	5	7	75	227	170
396	65	86	173	204	184	21	244	138	34	208	128	185	30	2	212	137
397	239	89	67	206	95	130	76	214	80	157	173	240	119	217	69	19
398	61	101	160	28	86	36	242	118	161	204	108	243	237	68	21	230
399	119	159	27	224	39	232	86	152	124	220	125	248	146	204	141	52
400	231	161	215	148	34	116	158	99	107	209	15	155	10	127	59	210

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
351								X		X		X	X		X	X			X				X	X	X	X	X	X			X	X		X	X	X	X		X	X	X
352				X	X	X		X	X		X	X	X	X			X	X		X		X	X					X	X	X	X		X		X				X	X	
353	X			X	X	X		X		X	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	X	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		X		X	X	X	X	X	X	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	X		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	X	X			X	X	X		X		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	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	X	X	X		X	X	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		X		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	X		X	X			X	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		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	X	X	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	X	X	X	X	X	X			X	X		X		X		X	X					X	X			X				X	X	X	X		X	X	X			X	
365	X			X	X	X		X			X		X			X		X	X				X		X		X		X	X			X	X			X	X			X
366		X	X	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	X					X		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		X	X	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		X	X	X		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	X		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		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				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		X	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		X		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			X		X	X	X	X	X	X	X	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			X		X	X	X	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								X	X	X	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	X	X	X		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		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	X	X	X		X		X			X	X	X			X	X		X	X		X	X		X		X	X	X	X					X		X	X	X	X		
381		X	X	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	X	X	X	X	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	X	X	X			X			X	X			X	X		X		X				X			X		X	X	X							X	X		X	X	X
384			X	X			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	X		X		X	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		X	X	X	X	X	X	X	X		X	X		X			X	X		X		X		X	X				X	X	X	X				X				X	
387				X		X		X		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	X	X	X		X	X		X	X			X		X	X	X		X	X	X		X			X		X	X				X			X			X		X	
389	X	X		X	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	X							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				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	X	X			X		X			X				X						X	X	X				X	X			X	X				X		X	X	X	X	
393				X			X	X	X	X			X		X	X	X	X			X	X	X		X	X						X	X	X	X		X		X	X	X
394	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	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		X	X	X	X	X		X	X	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	X	X			X		X			X	X	X	X			X	X		X	X	X		X		X		X		X	X	X		X		X				X		X
398	X		X	X			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	X	X	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	X	X	X	X	X		X	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**

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

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
451	X	X		X	X	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	X	X	X	X	X		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		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		X	X	X	X	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			X	X		X		X	X	X		X		X		X			X		X	X	X		X	X	X	X	X	X	X					X			X	X	
456	X					X	X							X	X		X	X			X	X	X			X	X	X	X	X	X			X	X				X		X
457	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	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	X			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			X		X	X				X	X	X	X	X		X		X	X	X		X	X		X				X	X		X	X			X			X	X	
461			X			X		X		X	X				X	X	X	X	X	X			X	X	X	X		X	X		X		X		X					X	X
462		X	X		X	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	X						X		X	X		X	X	X	X					X		X	X		X			X	X	X	X	X	X			X	X		X	X	
464			X		X	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	X	X	X		X	X	X			X		X	X			X		X	X		X			X	X	X	X					X	X			X	X	X	X		
466	X			X	X		X		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		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		X	X		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	X	X				X	X	X	X	X	X		X	X			X		X			X	X		X			X				X	X	X		X			X		
470	X	X	X	X	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		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						X	X		X				X	X	X		X		X		X		X			X	X		X	X				X	X	X		X	X	X	X
473	X	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		X	X		X	X		X		X	X	X		X		X		X	X	X	X	X	X								X	X	X			X		X	X		X
475	X	X	X	X	X	X	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	X	X			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			X	X				X		X	X		X	X		X	X		X			X	X	X	X		X	X			X		X	X				X	X		X
478	X	X	X		X	X			X					X				X	X	X			X		X		X			X			X	X	X	X	X	X	X	X	X
479	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		X	X		X	X			X		X	X				X	X			X	X		X			X				X		X	X	X	X	X	X	X	X		X
481		X			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	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	X	X			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	X		X	X	X	X		X		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	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	X	X	X	X			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	X	X		X	X			X	X			X	X					X			X	X	X		X	X	X	X	X	X				X	X	X				X	X
488	X		X	X	X	X	X		X			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	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	X		X				X	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	X	X		X	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	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		X			X	X	X				X	X		X	X	X	X	X		X		X	X	X			X			X			X					X	X	X	X
494		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	X	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				X	X	X	X	X			X			X		X		X	X	X		X		X	X		X	X	X								X	X	X	X	
497	X			X	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	X		X	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	X	X	X		X	X		X	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	X	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

[E2E0294] ⌈These requirements are not applicable to this specification.⌋ (BSW00338, BSW168, BSW00375, BSW00339, BSW00369, BSW00336, BSW00435)