# Specification of SW-C End-to-End Communication Protection Library

<table>
<thead>
<tr>
<th>Document Title</th>
<th>Specification of SW-C End-to-End Communication Protection Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Owner</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>Document Responsibility</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>Document Identification No</td>
<td>428</td>
</tr>
<tr>
<td>Document Status</td>
<td>Final</td>
</tr>
<tr>
<td>Part of AUTOSAR Standard</td>
<td>Classic Platform</td>
</tr>
<tr>
<td>Part of Standard Release</td>
<td>4.3.1</td>
</tr>
</tbody>
</table>

## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 2017-12-08 | 4.3.1   | AUTOSAR Release Management | • Updated traceability to SRS E2E.  
• Fixed enumeration literals for E2E_PxxCheckStatusType for profiles 1 and 2.  
• Corrected name of step E2E_SMClearProfileStatus to E2E_SMClearStatus in Routine E2E_SM_checkinit  
• Various clarifications in configuration and routine parameters, mainly of profile 2 and 7. |
| 2016-11-30 | 4.3.0   | AUTOSAR Release Management | • Added new Profiles 7, 11 and 22.  
• Fixed initialization of profile 1 and 2 in the init function. Now properly sets WaitForFirstData to TRUE.  
• Corrected/unified initialization of Counter state variable and bit/byte conversion in configuration data in profiles 4, 5, and 6.  
• Removed chapter 8.3.7 elementary protocol functions that were marked obsolete since several releases. |
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 2015-07-31 | 4.2.2   | AUTOSAR Release Management | • Introduced new E2E state machine profile status E2E_P_NONEWDATA. Adapted figures, API tables and mapping functions. This solves an issue with deterministic startup of the state machine.  
• Updated Figure 7-7, added behavior in case ReceivedCounter is out of range.  
• Assigned new specification ID SWS_E2E_00478 to duplicate specification SWS_E2E_00324 (specification of profile 4).  
• Fixed figure 7-6 “Calculate CRC over Data ID and Data”, which was already fixed in R4.1.2 but falsely included as of R4.1.1. |
| 2014-10-31 | 4.2.1   | AUTOSAR Release Management | • Introduction of E2E profiles 4, 5, 6  
• Introduction of E2E state machine  
• Introduction of init functions and status mapping functions for profiles 1, 2  
• Overview of wrapper, by means of several new diagrams. |
| 2014-03-31 | 4.1.3   | AUTOSAR Release Management | • Editorial changes |
| 2013-10-31 | 4.1.2   | AUTOSAR Release Management | • Correction in E2E variant 1C  
• Various minor corrections  
• Editorial changes |
| 2013-03-15 | 4.1.1   | AUTOSAR Release Management | • Full support for E2E protection at signal group level  
• Removed dependency to Rte_IsUpdated  
• Changed recommendations about the maximum data lengths  
• Addition of initialization functions to the redundant wrapper  
• Corrections in code examples |
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 2013-03-15 | 4.1.1   | AUTOSAR Administration | - Reworked according to the new SWS_BSWGeneral  
- New indexing scheme for requirements  
- Extension of E2E Profile 1 to support 12-bit Data IDs (variant 1C)  
- Alignment with ISO 26262 (terms, communication faults)  
- Quality ameliorations (due to document review)  
- Clarification in the configuration of E2E parameters |
| 2011-12-22 | 4.0.3   | AUTOSAR Administration | - E2E Profile 3 removed (not backward compatible)  
- Several bugfixes in of E2E Protection Wrapper API (not backward compatible)  
- Modified return values of E2E Protection Wrapper API (not backward compatible)  
- Addition of init API for the E2E Protection Wrapper  
- Several bugfixes and modifications in code examples of E2E Protection Wrapper  
- Extensions in configuration, making sender and receiver more independent  
- Bugfix in the profile 1 alternating mode CRC calculation  
- Clarifications with in E2E Profile 1 with respect to the CRC  
- Several minor bug fixes  
- Several optimizations in the text descriptions  
- New template with requirements traceability |
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-09-30</td>
<td>3.1.5</td>
<td>AUTOSAR Administration</td>
<td>• Corrected the wrapper configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Corrected the code example for the usage of the wrapper.</td>
</tr>
<tr>
<td>2010-02-02</td>
<td>3.1.4</td>
<td>AUTOSAR Administration</td>
<td>• Initial Release</td>
</tr>
</tbody>
</table>
Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.
# Table of Contents

1 Introduction and functional overview ................................................................. 10
2 Acronyms and abbreviations ................................................................................. 12
3 Related documentation ......................................................................................... 13
   3.1 Input documents ............................................................................................. 13
   3.2 Related standards and norms ......................................................................... 14
4 Constraints and assumptions ................................................................................ 15
   4.1 Limitations ...................................................................................................... 15
   4.1.1 Limitations when invoking library at the level of data elements ................. 15
   4.2 Applicability to automotive domains .............................................................. 16
   4.3 Background information concerning functional safety ................................. 16
      4.3.1 Functional safety and communication .................................................. 16
      4.3.2 Sources of faults in E2E communication ............................................. 17
      4.3.3 Communication faults .......................................................................... 18
   4.4 Implementation of the E2E Library ................................................................. 19
5 Dependencies to/from other modules ................................................................. 20
   5.1.1 Required file structure .............................................................................. 20
   5.1.2 Dependency on CRC library ...................................................................... 21
6 Requirements traceability .................................................................................... 22
7 Functional specification ......................................................................................... 38
   7.1 Overview of communication protection ......................................................... 38
   7.2 Overview of E2E Profiles .............................................................................. 39
      7.2.1 Error classification ............................................................................... 40
      7.2.2 Error detection ...................................................................................... 41
   7.3 Specification of E2E Profile 1 .......................................................................... 41
      7.3.1 Data Layout ............................................................................................ 43
      7.3.2 Counter ................................................................................................... 43
      7.3.3 Data ID .................................................................................................. 44
      7.3.4 CRC calculation ..................................................................................... 45
      7.3.5 Timeout detection .................................................................................. 46
      7.3.6 E2E Profile 1 variants ............................................................................ 46
      7.3.7 E2E_P01Protect .................................................................................... 47
      7.3.8 Calculate CRC ...................................................................................... 48
      7.3.9 E2E_P01Check ...................................................................................... 50
   7.4 Specification of E2E Profile 2 .......................................................................... 52
      7.4.1 E2E_P02Protect .................................................................................... 54
      7.4.2 E2E_P02Check ...................................................................................... 56
   7.5 Specification of E2E Profile 4 .......................................................................... 63
      7.5.1 Data Layout ............................................................................................ 64
      7.5.2 Counter ................................................................................................... 64
      7.5.3 Data ID .................................................................................................. 65
      7.5.4 Length ................................................................................................... 65
      7.5.5 CRC ....................................................................................................... 66
      7.5.6 Timeout detection .................................................................................. 66
      7.5.7 E2E Profile 4 variants ............................................................................ 66
7.5.8 E2E_P04Protect ................................................................. 66
7.5.9 E2E_P04Check ............................................................... 71
7.6 Specification of E2E Profile 5 .............................................. 76
  7.6.1 Data Layout ............................................................... 76
  7.6.2 Counter ................................................................. 77
  7.6.3 Data ID ................................................................. 77
  7.6.4 Length ........................................................................ 78
  7.6.5 CRC .......................................................................... 78
  7.6.6 Timeout detection ....................................................... 78
  7.6.7 E2E_P05Protect .......................................................... 79
  7.6.8 E2E_P05Check ........................................................... 84
7.7 Specification of E2E Profile 6 .............................................. 87
  7.7.1 Data Layout ............................................................... 88
  7.7.2 Counter ................................................................. 88
  7.7.3 Data ID ................................................................. 89
  7.7.4 Length ........................................................................ 90
  7.7.5 CRC .......................................................................... 90
  7.7.6 Timeout detection ....................................................... 90
  7.7.7 E2E_P06Protect .......................................................... 90
  7.7.8 E2E_P06Check ........................................................... 96
7.8 Specification of E2E Profile 7 .............................................. 100
  7.8.1 Data Layout ............................................................... 101
  7.8.2 Counter ................................................................. 102
  7.8.3 Data ID ................................................................. 102
  7.8.4 Length ........................................................................ 103
  7.8.5 CRC .......................................................................... 103
  7.8.6 Timeout detection ....................................................... 103
  7.8.7 E2E Profile 7 variants ................................................ 103
  7.8.8 E2E_P07Protect .......................................................... 103
  7.8.9 E2E_P07Check ........................................................... 108
7.9 Specification of E2E Profile 11 ........................................... 113
  7.9.1 Data Layout ............................................................... 113
  7.9.2 Counter ................................................................. 115
  7.9.3 Data ID ................................................................. 115
  7.9.4 Length ........................................................................ 116
  7.9.5 CRC .......................................................................... 116
  7.9.6 Timeout detection ....................................................... 116
  7.9.7 E2E_P11Protect .......................................................... 117
  7.9.8 E2E_P11Check ........................................................... 121
7.10 Specification of E2E Profile 22 ......................................... 126
  7.10.1 Data Layout ............................................................... 126
  7.10.2 Counter ................................................................. 127
  7.10.3 Data ID List ............................................................. 128
  7.10.4 Length ...................................................................... 128
  7.10.5 CRC ........................................................................ 128
  7.10.6 Timeout detection ....................................................... 129
  7.10.7 E2E_P22Protect ........................................................ 129
  7.10.8 E2E_P22Check .......................................................... 132
7.11 Specification of E2E state machine .................................... 137
  7.11.1 Overview of the state machine .................................... 137
7.11.2 State machine specification ......................................................... 137
7.12 Version Check .................................................................................. 141
8 API specification .................................................................................. 143
  8.1 Imported types .................................................................................. 143
  8.2 Type definitions ................................................................................ 143
      8.2.1 E2E Profile 1 types .................................................................... 144
      8.2.2 E2E Profile 2 types .................................................................... 149
      8.2.3 E2E Profile 4 types .................................................................... 153
      8.2.4 E2E Profile 5 types .................................................................... 155
      8.2.5 E2E Profile 6 types .................................................................... 159
      8.2.6 E2E Profile 7 types .................................................................... 163
      8.2.7 E2E Profile 11 types .................................................................... 166
      8.2.8 E2E Profile 22 types .................................................................... 170
      8.2.9 E2E state machine types ............................................................. 173
  8.3 Routine definitions ............................................................................. 176
      8.3.1 E2E Profile 1 routines ................................................................ 176
      8.3.2 E2E Profile 2 routines ................................................................ 179
      8.3.3 E2E Profile 4 routines ................................................................ 184
      8.3.4 E2E Profile 5 routines ................................................................ 186
      8.3.5 E2E Profile 6 routines ................................................................ 189
      8.3.6 E2E Profile 7 routines ................................................................ 192
      8.3.7 E2E Profile 11 routines ............................................................... 196
      8.3.8 E2E Profile 22 routines ............................................................... 199
      8.3.9 E2E State machine routines ......................................................... 202
      8.3.10 Auxiliary Functions .................................................................. 203
  8.4 Call-back notifications ....................................................................... 204
  8.5 Scheduled functions .......................................................................... 204
  8.6 Expected Interfaces .......................................................................... 204
      8.6.1 Mandatory Interfaces ................................................................. 204
9 Sequence Diagrams for invoking E2E Library ........................................ 205
    9.1 Sender ......................................................................................... 205
      9.1.1 Sender of data elements ............................................................ 205
      9.1.2 Sender at signal group level ....................................................... 207
    9.2 Receiver ....................................................................................... 208
      9.2.1 Receiver at data element level ................................................... 210
      9.2.2 Receiver at signal group level .................................................. 212
10 Configuration specification ................................................................. 214
  10.1 Published Information ..................................................................... 214
    11.1 E2E profiles and their standard variants ...................................... 215
    11.2 E2E error handling ...................................................................... 215
    11.3 Maximal lengths of Data, communication buses ........................... 215
    11.4 Methodology of usage of E2E Library .......................................... 217
    11.5 Configuration constraints on Data IDs .......................................... 218
      11.5.1 Data IDs .............................................................................. 218
      11.5.2 Double Data ID configuration of E2E Profile 1 ...................... 218
11.5.3 Alternating Data ID configuration of E2E Profile 1 ........................................ 219
11.5.4 Nibble configuration of E2E Profile 1 .................................................................. 220
11.6 Building custom E2E protocols ............................................................................. 220
11.7 I-PDU Layout ......................................................................................................... 221
11.7.1 Alignment of signals to byte limits .................................................................... 221
11.7.2 Unused bits .......................................................................................................... 221
11.7.3 Byte order (Endianness) .................................................................................... 222
11.7.4 Bit order .............................................................................................................. 224
11.8 RTE configuration constraints for SW-C level protection ..................................... 225
11.8.1 Communication model for SW-C level protection .............................................. 225
11.8.2 Multiplicities for SW-C level protection ............................................................. 225
11.8.3 Explicit access .................................................................................................... 225
11.9 Restrictions on the use of COM features ............................................................... 226
11.10 Examples for the implementation of E2E protection concepts based on E2E-Library - Branch ................................................................. 228
11.10.1 Basic principles ................................................................................................ 228
11.10.2 Determination of the integrity of a communication channel within the receiver ................................................................. 229

12 Annex B: Application hints on usage of E2E Library ................................................ 232
12.1 E2E Protection Wrapper ......................................................................................... 233
12.1.1 Functional overview ......................................................................................... 234
12.1.2 Application scenario with Transmission Manager ............................................ 235
12.1.3 Application scenario with E2E Manager and Conversion Manager ................. 237
12.1.4 File structure .................................................................................................... 240
12.1.5 Methodology ..................................................................................................... 242
12.1.6 Error classification ......................................................................................... 246
12.1.7 E2E Protection Wrapper routines ..................................................................... 248
12.1.8 E2EPW Routines Diagrams ............................................................................. 263
12.1.9 Code Example ................................................................................................. 274
12.2 COM E2E Callouts ............................................................................................... 293
12.2.1 Functional overview ....................................................................................... 293
12.2.2 Methodology .................................................................................................. 297
12.2.3 Code Example ............................................................................................... 299
12.3 Provision of the Protection Wrapper Interface on a ECU with COM Callout solution ................................................................................................. 300
12.4 Protection at RTE level through E2E Transformer ................................................. 300

13 Usage and generation of DataIDLists for E2E profile 2 ......................................... 301
13.1 Example A (persistent routing error) .................................................................... 301
13.2 Example B (forbidden configuration) ................................................................... 303
13.3 Conclusion ............................................................................................................. 304
13.4 DataIDList example ............................................................................................ 304

14 Not applicable requirements ...................................................................................... 315
1 Introduction and functional overview

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

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

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

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

The E2E protection allows the following:

1. It protects the safety-related data elements to be sent over the RTE by attaching control data,
2. It verifies the safety-related data elements received from the RTE using this control data, and
3. It indicates that received safety-related data elements faulty, which then has to be handled by the receiver SW-C.

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

The E2E library is invoked from:
1. E2E Transformer (a new, standardized way to invoke E2E, introduced in R4.2.1)
2. E2E Protection Wrapper
3. COM E2E Callout.

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Abbreviation / Acronym:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E Library</td>
<td>Short name for the End-to-End Communication Protection Library</td>
</tr>
<tr>
<td>Data ID</td>
<td>An identifier that uniquely identifies the message / data element / data.</td>
</tr>
<tr>
<td>Repetition</td>
<td>Repetition of information (see 4.3.3.1)</td>
</tr>
<tr>
<td>Loss</td>
<td>Loss of information (see 4.3.3.2)</td>
</tr>
<tr>
<td>Delay</td>
<td>Delay of information (see 4.3.3.3)</td>
</tr>
<tr>
<td>Insertion</td>
<td>Insertion of information (see 4.3.3.4)</td>
</tr>
<tr>
<td>Masquerade</td>
<td>Masquerade (see 4.3.3.5)</td>
</tr>
<tr>
<td>Incorrect addressing</td>
<td>Incorrect addressing of information (see 4.3.3.6).</td>
</tr>
<tr>
<td>Incorrect sequence</td>
<td>Incorrect sequence of information (see 4.3.3.7).</td>
</tr>
<tr>
<td>Corruption</td>
<td>Corruption of information (see 4.3.3.8).</td>
</tr>
<tr>
<td>Asymmetric information</td>
<td>Asymmetric information sent from a sender to multiple receivers (see 4.3.3.9)</td>
</tr>
<tr>
<td>Subset</td>
<td>Information from a sender received by only a subset of the receivers (see 4.3.3.10)</td>
</tr>
<tr>
<td>Blocking</td>
<td>Blocking access to a communication channel (see 4.3.3.11)</td>
</tr>
</tbody>
</table>

Table 2-1: Acronyms and abbreviations

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

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

AUTOSAR_SRS_BSWGeneral.pdf

AUTOSAR_SWS_COM.pdf

AUTOSAR_SWS_Scheduler.pdf

AUTOSAR_SWS_MemoryMapping.pdf

AUTOSAR_SWS_CRCLibrary.pdf

AUTOSAR_SWS_PlatformTypes.pdf

[9] Requirements on Libraries
AUTOSAR_SRS_Libraries.pdf

[10] AUTOSAR Glossary
AUTOSAR_TR_Glossary.pdf

AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[12] System Template
AUTOSAR_TPS_SystemTemplate.pdf

AUTOSAR_TPS_ECUConfiguration.pdf
3.2 Related standards and norms

http://www.iso.org/
4 Constraints and assumptions

4.1 Limitations

E2E Profile 2 has in R4.2.1 a new setting offset. This offset can be configured in the system template. However, the E2E Profile 2 specification does not support the case when offset is different than 0. The specification of E2E Profile 2 will be fixed in a future AUTOSAR release, to support a configurable offset.

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

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

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

4.1.1 Limitations when invoking library at the level of data elements

[UC_E2E_00224] If the E2E Library is invoked at the level of data elements (e.g. from SW-Cs or from E2E Protection Wrapper), then the communication shall be an explicit sender-receiver communication, in 1:1 and 1:N multiplicities. (SRS_E2E_08528)

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

[UC_E2E_00255] If the E2E Library is invoked at the level of data elements and 1:N communication model is used and the data elements are sent using more than one I-PDU, then all these I-PDUs shall have the same layout. (SRS_E2E_08528)
[UC_E2E_00226] For each 1:N sender-receiver relationship the user of AUTOSAR shall define one specific layout to which the data elements that are going be protected by E2E-Library are mapped for data transmission.  (SRS_E2E_08528)

[UC_E2E_00326] In case a user of AUTOSAR needs protected intra-ECU communication and protected inter-ECU communication to implement a safety-related sender-receiver relationship, the defined inter-ECU communication I-PDU layout shall be used for both transmissions.  (SRS_E2E_08528)

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

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

4.2 Applicability to automotive domains

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

4.3 Background information concerning functional safety

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

4.3.1 Functional safety and communication

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

- repetition of information;
- loss of information;
4.3.2 Sources of faults in E2E communication

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

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

These three sources are described in the sections below.

4.3.2.1 Software faults

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

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

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

4.3.2.2 Random hardware faults

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

4.3.2.3 External influences, environmental stress

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

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

4.3.3.1 Repetition of information

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

4.3.3.2 Loss of information

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

4.3.3.3 Delay of information

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

4.3.3.4 Insertion of information

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

4.3.3.5 Masquerading

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

4.3.3.6 Incorrect addressing

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

4.3.3.7 Incorrect sequence of information

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

4.3.3.8 Corruption of information

A type of communication fault, which changes information.
4.3.3.9 Asymmetric information sent from a sender to multiple receivers

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

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

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

4.3.3.11 Blocking access to a communication channel

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

4.4 Implementation of the E2E Library

[SWS_E2E_00050] The implementation of the E2E Library shall comply with the requirements for the development of safety-related software for the automotive domain. (SRS_E2E_08527)

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

[SWS_E2E_00311] The configuration of the E2E Library and of the code invoking it (e.g. E2E wrapper or E2E callouts) shall be implemented and configured (including configuration options used from other subsystems, e.g. COM signal to I-PDU mapping) according to the requirements for the development of safety-related software for the automotive domain. (SRS_E2E_08528)
5 Dependencies to/from other modules

5.1.1 Required file structure

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

![Diagram of file dependencies](image)

**Figure 5-1: File dependencies**

[SWS_E2E_00048]** E2E library shall be built of the following files: E2E.h (common header), E2E.c (implementation of common parts), E2E_PXX.c and E2E_PXX.h (where XX: e.g. 01, 02, … representing the profile) and E2E_SM.c and E2E_SM.h (for E2E state machine).** (SRS_E2E_08528)

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

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

[SWS_E2E_00115]** E2E library files (i.e. E2E_.*) shall not include any RTE files.** (SRS_E2E_08528)
Note that as there are no configuration options in the E2E library, there is no E2E_Cfg.h file. Moreover, ComStack_Types.h are not needed by E2E, neither are RTE header files.

5.1.2 Dependency on CRC library

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

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

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

To have the same results of the functions E2E_P01Protect() and E2E_P02Check() in >=R4.0 and R3.2, while using differently functioning CRC library, E2E “compensates” different behavior of the CRC library. This results with different invocation of the CRC library by E2E library (see Figure 7-6) in >=R4.0 and R3.2.
## 6 Requirements traceability

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Satisfied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRS_BSW_00003</td>
<td>All software modules shall provide version and identification information</td>
<td>SWS_E2E_00032, SWS_E2E_00327,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SWS_E2E_00467</td>
</tr>
<tr>
<td>SRS_BSW_00004</td>
<td>All Basic SW Modules shall perform a pre-processor check of the versions of all imported include files</td>
<td>SWS_E2E_00038</td>
</tr>
<tr>
<td>SRS_BSW_00005</td>
<td>Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00006</td>
<td>The source code of software modules above the μC Abstraction Layer (MCAL) shall not be processor and compiler dependent.</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00007</td>
<td>All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00009</td>
<td>All Basic SW Modules shall be documented according to a common standard.</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00010</td>
<td>The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00101</td>
<td>The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function.</td>
<td>SWS_E2E_00037</td>
</tr>
<tr>
<td>SRS_BSW_00158</td>
<td>All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00159</td>
<td>All modules of the AUTOSAR Basic Software shall support a tool based configuration</td>
<td>SWS_E2E_00037</td>
</tr>
<tr>
<td>SRS_BSW_00160</td>
<td>Configuration files of AUTOSAR Basic SW module shall be readable for human beings</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00161</td>
<td>The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00162</td>
<td>The AUTOSAR Basic Software shall provide a hardware</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>Specification</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00164</td>
<td>The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00167</td>
<td>All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00168</td>
<td>SW components shall be tested by a function defined in a common API in the Basis-SW</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00170</td>
<td>The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00171</td>
<td>Optional functionality of a Basic-SW component that is not required in the ECU shall be configurable at pre-compile-time</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00172</td>
<td>The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00300</td>
<td>All AUTOSAR Basic Software Modules shall be identified by an unambiguous name</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00301</td>
<td>All AUTOSAR Basic Software Modules shall only import the necessary information</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00302</td>
<td>All AUTOSAR Basic Software Modules shall only export information needed by other modules</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00304</td>
<td>All AUTOSAR Basic Software Modules shall use the following data types instead of native C data types</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00305</td>
<td>Data types naming convention</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00306</td>
<td>AUTOSAR Basic Software Modules shall be compiler and platform independent</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00307</td>
<td>Global variables naming convention</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00308</td>
<td>AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file</td>
<td></td>
</tr>
<tr>
<td>Specification ID</td>
<td>Description</td>
<td>Reference</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>SRS_BSW_00309</td>
<td>All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00310</td>
<td>API naming convention</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00312</td>
<td>Shared code shall be reentrant</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00314</td>
<td>All internal driver modules shall separate the interrupt frame definition from the service routine</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00318</td>
<td>Each AUTOSAR Basic Software Module file shall provide version numbers in the header file</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00321</td>
<td>The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00323</td>
<td>All AUTOSAR Basic Software Modules shall check passed API parameters for validity</td>
<td>SWS_E2E 00047</td>
</tr>
<tr>
<td>SRS_BSW_00325</td>
<td>The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00327</td>
<td>Error values naming convention</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00328</td>
<td>All AUTOSAR Basic Software Modules shall avoid the duplication of code</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00330</td>
<td>It shall be allowed to use macros instead of functions where source code is used and runtime is critical</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00331</td>
<td>All Basic Software Modules shall strictly separate error and status information</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00333</td>
<td>For each callback function it shall be specified if it is called from interrupt context or not</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00334</td>
<td>All Basic Software Modules shall provide an XML file that contains the meta data</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00335</td>
<td>Status values naming convention</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00336</td>
<td>Basic SW module shall be able to shutdown</td>
<td>SWS_E2E NA_00294</td>
</tr>
<tr>
<td>SRS_BSW_00337</td>
<td>Classification of development errors</td>
<td>SWS_E2E 00047</td>
</tr>
<tr>
<td>SRS_BSW_00339</td>
<td>Reporting of production relevant</td>
<td>SWS_E2E 00216, SWS_E2E NA_00294</td>
</tr>
<tr>
<td>Specification</td>
<td>Module Documentation Requirements</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00341</td>
<td>Module documentation shall contain all needed informations</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00342</td>
<td>It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00343</td>
<td>The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00344</td>
<td>BSW Modules shall support link-time configuration</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00345</td>
<td>BSW Modules shall support pre-compile configuration</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00346</td>
<td>All AUTOSAR Basic Software Modules shall provide at least a basic set of module files</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00347</td>
<td>A Naming separation of different instances of BSW drivers shall be in place</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00348</td>
<td>All AUTOSAR standard types and constants shall be placed and organized in a standard type header file</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00350</td>
<td>All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00351</td>
<td>Encapsulation of compiler specific methods to map objects</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00353</td>
<td>All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00357</td>
<td>For success/failure of an API call a standard return type shall be defined</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00358</td>
<td>The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00359</td>
<td>All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00360</td>
<td>AUTOSAR Basic Software</td>
<td></td>
</tr>
<tr>
<td>Specification of SW-C End-to-End Communication Protection Library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUTOSAR CP Release 4.3.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>SRS/BSE Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules callback functions are allowed to have parameters</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header</td>
<td></td>
</tr>
<tr>
<td>All AUTOSAR Basic Software Modules shall not return specific development error codes via the API</td>
<td>SWS_E2E_00049, SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>The passing of function pointers as API parameter is forbidden for all AUTOSAR Basic Software Modules</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>All Basic Software Modules shall provide a readable module vendor identification</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>Basic Software Modules shall report wake-up reasons</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>A Basic Software Module can return a module specific types</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>AUTOSAR shall provide a boolean type</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>All software modules shall provide a module identifier in the header file and in the module XML description file.</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>Configuration parameters being stored in memory shall be placed into separate c-files</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>The pre-compile time parameters shall be placed into a separate configuration header file</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>The Basic Software Module specifications shall specify at least in the description which other modules they require</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>List possible error notifications</td>
<td>SWS_E2E_NA_00294</td>
</tr>
<tr>
<td>Specification</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SRS_BSW_00386</td>
<td>The BSW shall specify the configuration for detecting an error</td>
</tr>
<tr>
<td>SRS_BSW_00388</td>
<td>Containers shall be used to group configuration parameters that are defined for the same object</td>
</tr>
<tr>
<td>SRS_BSW_00389</td>
<td>Containers shall have names</td>
</tr>
<tr>
<td>SRS_BSW_00390</td>
<td>Parameter content shall be unique within the module</td>
</tr>
<tr>
<td>SRS_BSW_00392</td>
<td>Parameters shall have a type</td>
</tr>
<tr>
<td>SRS_BSW_00393</td>
<td>Parameters shall have a range</td>
</tr>
<tr>
<td>SRS_BSW_00394</td>
<td>The Basic Software Module specifications shall specify the scope of the configuration parameters</td>
</tr>
<tr>
<td>SRS_BSW_00395</td>
<td>The Basic Software Module specifications shall list all configuration parameter dependencies</td>
</tr>
<tr>
<td>SRS_BSW_00396</td>
<td>The Basic Software Module specifications shall specify the supported configuration classes for changing values and multiplicities for each parameter/container</td>
</tr>
<tr>
<td>SRS_BSW_00397</td>
<td>The configuration parameters in pre-compile time are fixed before compilation starts</td>
</tr>
<tr>
<td>SRS_BSW_00398</td>
<td>The link-time configuration is achieved on object code basis in the stage after compiling and before linking</td>
</tr>
<tr>
<td>SRS_BSW_00399</td>
<td>Parameter-sets shall be located in a separate segment and shall be loaded after the code</td>
</tr>
<tr>
<td>SRS_BSW_00400</td>
<td>Parameter shall be selected from multiple sets of parameters after code has been loaded and started</td>
</tr>
<tr>
<td>SRS_BSW_00401</td>
<td>Documentation of multiple instances of configuration parameters shall be available</td>
</tr>
<tr>
<td>SRS_BSW_00402</td>
<td>Each module shall provide version information</td>
</tr>
<tr>
<td>SRS_BSW_00403</td>
<td>The Basic Software Module specifications shall specify for each parameter/container whether it supports different values or multiplicity in different configuration sets</td>
</tr>
<tr>
<td>Specification ID</td>
<td>Requirement</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SRS_BSW_00404</td>
<td>BSW Modules shall support post-build configuration</td>
</tr>
<tr>
<td>SRS_BSW_00405</td>
<td>BSW Modules shall support multiple configuration sets</td>
</tr>
<tr>
<td>SRS_BSW_00406</td>
<td>A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called</td>
</tr>
<tr>
<td>SRS_BSW_00407</td>
<td>Each BSW module shall provide a function to read out the version information of a dedicated module implementation</td>
</tr>
<tr>
<td>SRS_BSW_00408</td>
<td>All AUTOSAR Basic Software Modules configuration parameters shall be named according to a specific naming rule</td>
</tr>
<tr>
<td>SRS_BSW_00409</td>
<td>All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration</td>
</tr>
<tr>
<td>SRS_BSW_00410</td>
<td>Compiler switches shall have defined values</td>
</tr>
<tr>
<td>SRS_BSW_00411</td>
<td>All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API</td>
</tr>
<tr>
<td>SRS_BSW_00412</td>
<td>References to c-configuration parameters shall be placed into a separate h-file</td>
</tr>
<tr>
<td>SRS_BSW_00413</td>
<td>An index-based accessing of the instances of BSW modules shall be done</td>
</tr>
<tr>
<td>SRS_BSW_00414</td>
<td>Init functions shall have a pointer to a configuration structure as single parameter</td>
</tr>
<tr>
<td>SRS_BSW_00415</td>
<td>Interfaces which are provided exclusively for one module shall be separated into a dedicated header file</td>
</tr>
<tr>
<td>SRS_BSW_00416</td>
<td>The sequence of modules to be initialized shall be configurable</td>
</tr>
<tr>
<td>SRS_BSW_00417</td>
<td>Software which is not part of the SW-C shall report error events only after the DEM is fully operational.</td>
</tr>
<tr>
<td>SRS_BSW_00419</td>
<td>If a pre-compile time configuration parameter is</td>
</tr>
<tr>
<td>Specification Item</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>implemented as &quot;const&quot; it should be placed into a separate c-file</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00422</td>
<td>Pre-de-bouncing of error status information is done within the DEM</td>
</tr>
<tr>
<td>SRS_BSW_00423</td>
<td>BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template</td>
</tr>
<tr>
<td>SRS_BSW_00424</td>
<td>BSW module main processing functions shall not be allowed to enter a wait state</td>
</tr>
<tr>
<td>SRS_BSW_00425</td>
<td>The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects</td>
</tr>
<tr>
<td>SRS_BSW_00426</td>
<td>BSW Modules shall ensure data consistency of data which is shared between BSW modules</td>
</tr>
<tr>
<td>SRS_BSW_00427</td>
<td>ISR functions shall be defined and documented in the BSW module description template</td>
</tr>
<tr>
<td>SRS_BSW_00428</td>
<td>A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence</td>
</tr>
<tr>
<td>SRS_BSW_00429</td>
<td>Access to OS is restricted</td>
</tr>
<tr>
<td>SRS_BSW_00432</td>
<td>Modules should have separate main processing functions for read/receive and write/transmit data path</td>
</tr>
<tr>
<td>SRS_BSW_00433</td>
<td>Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler</td>
</tr>
<tr>
<td>SRS_BSW_00437</td>
<td>Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup</td>
</tr>
<tr>
<td>SRS_BSW_00438</td>
<td>Configuration data shall be defined in a structure</td>
</tr>
<tr>
<td>SRS_BSW_00439</td>
<td>Enable BSW modules to handle interrupts</td>
</tr>
<tr>
<td>SRS_BSW_00440</td>
<td>The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API</td>
</tr>
<tr>
<td>SRS_BSW_00441</td>
<td>Naming convention for type,</td>
</tr>
<tr>
<td>Specification</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SRS_BSW_00447</td>
<td>Standardizing Include file structure of BSW Modules</td>
</tr>
<tr>
<td>SRS_BSW_00448</td>
<td>Module SWS shall not contain requirements from Other Modules</td>
</tr>
<tr>
<td>SRS_BSW_00449</td>
<td>BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType</td>
</tr>
<tr>
<td>SRS_BSW_00450</td>
<td>A Main function of a un-initialized module shall return immediately</td>
</tr>
<tr>
<td>SRS_BSW_00451</td>
<td>Hardware registers shall be protected if concurrent access to these registers occur</td>
</tr>
<tr>
<td>SRS_BSW_00452</td>
<td>Classification of runtime errors</td>
</tr>
<tr>
<td>SRS_BSW_00453</td>
<td>BSW Modules shall be harmonized</td>
</tr>
<tr>
<td>SRS_BSW_00454</td>
<td>An alternative interface without a parameter of category DATA_REFERENCE shall be available.</td>
</tr>
<tr>
<td>SRS_BSW_00456</td>
<td>A Header file shall be defined in order to harmonize BSW Modules</td>
</tr>
<tr>
<td>SRS_BSW_00457</td>
<td>Callback functions of Application software components shall be invoked by the Basis SW</td>
</tr>
<tr>
<td>SRS_BSW_00458</td>
<td>Classification of production errors</td>
</tr>
<tr>
<td>SRS_BSW_00459</td>
<td>It shall be possible to concurrently execute a service offered by a BSW module in different partitions</td>
</tr>
<tr>
<td>SRS_BSW_00460</td>
<td>Reentrancy Levels</td>
</tr>
<tr>
<td>SRS_BSW_00461</td>
<td>Modules called by generic modules shall satisfy all interfaces requested by the generic module</td>
</tr>
<tr>
<td>SRS_BSW_00462</td>
<td>All Standardized Autosar Interfaces shall have unique requirement Id / number</td>
</tr>
<tr>
<td>SRS_BSW_00463</td>
<td>Naming convention of callout prototypes</td>
</tr>
<tr>
<td>SRS_BSW_00464</td>
<td>File names shall be considered case sensitive regardless of the filesystem in which they are used</td>
</tr>
<tr>
<td>SRS_BSW_00465</td>
<td>It shall not be allowed to name any two files so that they only differ by the cases of their letters</td>
</tr>
<tr>
<td>SRS_BSW_00466</td>
<td>Classification of extended production errors</td>
</tr>
<tr>
<td>SRS_BSW_00467</td>
<td>The init / deinit services shall only be called by BsmM or EcUM</td>
</tr>
<tr>
<td>SRS_BSW_00469</td>
<td>Fault detection and healing of production errors and extended production errors</td>
</tr>
<tr>
<td>SRS_BSW_00470</td>
<td>Execution frequency of production error detection</td>
</tr>
<tr>
<td>SRS_BSW_00471</td>
<td>Do not cause dead-locks on detection of production errors - the ability to heal from previously detected production errors</td>
</tr>
<tr>
<td>SRS_BSW_00472</td>
<td>Avoid detection of two production errors with the same root cause.</td>
</tr>
<tr>
<td>SRS_BSW_00473</td>
<td>Classification of transient faults</td>
</tr>
<tr>
<td>SRS_BSW_00477</td>
<td>The functional interfaces of AUTOSAR BSW modules shall be specified in C90</td>
</tr>
<tr>
<td>SRS_BSW_00478</td>
<td>Timing limits of main functions</td>
</tr>
<tr>
<td>SRS_BSW_00479</td>
<td>Interfaces for handling request from external devices</td>
</tr>
<tr>
<td>SRS_BSW_00480</td>
<td>NullPointer Errors shall follow a naming rule</td>
</tr>
<tr>
<td>SRS_BSW_00481</td>
<td>Invalid configuration set selection errors shall follow a naming rule</td>
</tr>
<tr>
<td>SRS_BSW_00482</td>
<td>Get Version Information function shall follow a naming rule</td>
</tr>
<tr>
<td>SRS_BSW_00483</td>
<td>BSW Modules shall handle buffer alignments internally</td>
</tr>
<tr>
<td>SRS_E2E_08527</td>
<td>E2E library shall provide E2E profiles, in a form of library functions</td>
</tr>
<tr>
<td>Specification of SW-C End-to-End Communication Protection Library</td>
<td>AUTOSAR CP Release 4.3.1</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td><strong>SRS_E2E_08528</strong></td>
<td>E2E library shall provide E2E profiles, where each E2E profile completely defines a particular safety protocol</td>
</tr>
<tr>
<td>SWS_E2E_00011, SWS_E2E_00012, SWS_E2E_00017, SWS_E2E_00018, SWS_E2E_00020, SWS_E2E_00021, SWS_E2E_00033, SWS_E2E_00048, SWS_E2E_00075, SWS_E2E_00076, SWS_E2E_00085, SWS_E2E_00110, SWS_E2E_00115, SWS_E2E_00118, SWS_E2E_00119, SWS_E2E_00120, SWS_E2E_00121, SWS_E2E_00122, SWS_E2E_00123, SWS_E2E_00124, SWS_E2E_00125, SWS_E2E_00126, SWS_E2E_00127, SWS_E2E_00128, SWS_E2E_00129, SWS_E2E_00130, SWS_E2E_00132, SWS_E2E_00133, SWS_E2E_00134, SWS_E2E_00135, SWS_E2E_00136, SWS_E2E_00137, SWS_E2E_00138, SWS_E2E_00139, SWS_E2E_00140, SWS_E2E_00141, SWS_E2E_00142, SWS_E2E_00143, SWS_E2E_00145, SWS_E2E_00146, SWS_E2E_00147, SWS_E2E_00148, SWS_E2E_00149, SWS_E2E_00150, SWS_E2E_00151, SWS_E2E_00152, SWS_E2E_00153, SWS_E2E_00154, SWS_E2E_00155, SWS_E2E_00156, SWS_E2E_00157, SWS_E2E_00158, SWS_E2E_00159, SWS_E2E_00160, SWS_E2E_00161, SWS_E2E_00162, SWS_E2E_00163, SWS_E2E_00164, SWS_E2E_00165, SWS_E2E_00166, SWS_E2E_00167, SWS_E2E_00168, SWS_E2E_00169, SWS_E2E_00170, SWS_E2E_00171, SWS_E2E_00172, SWS_E2E_00173, SWS_E2E_00174, SWS_E2E_00175, SWS_E2E_00176, SWS_E2E_00177, SWS_E2E_00178, SWS_E2E_00179, SWS_E2E_00180, SWS_E2E_00181, SWS_E2E_00182, SWS_E2E_00183, SWS_E2E_00184, SWS_E2E_00185, SWS_E2E_00186, SWS_E2E_00187, SWS_E2E_00188, SWS_E2E_00189, SWS_E2E_00190, SWS_E2E_00191, SWS_E2E_00192, SWS_E2E_00193, SWS_E2E_00194, SWS_E2E_00195, SWS_E2E_00196, SWS_E2E_00200, SWS_E2E_00215, SWS_E2E_00217, SWS_E2E_00227, SWS_E2E_00228, SWS_E2E_00298, SWS_E2E_00299, SWS_E2E_00300, SWS_E2E_00301, SWS_E2E_00306, SWS_E2E_00307, SWS_E2E_00311, SWS_E2E_00314, SWS_E2E_00318, SWS_E2E_00319, SWS_E2E_00320, SWS_E2E_00321, SWS_E2E_00322, SWS_E2E_00323, SWS_E2E_00324, SWS_E2E_00325, SWS_E2E_00379, SWS_E2E_00380, SWS_E2E_00381, SWS_E2E_00382, SWS_E2E_00383, SWS_E2E_00384, SWS_E2E_00385, SWS_E2E_00386, SWS_E2E_00387, SWS_E2E_00388, SWS_E2E_00389, SWS_E2E_00390, SWS_E2E_00391, SWS_E2E_00392, SWS_E2E_00476, SWS_E2E_00477, SWS_E2E_00540, SWS_E2E_00541, SWS_E2E_00568, SWS_E2E_00570, SWS_E2E_00571, UC_E2E_00051, UC_E2E_00053, UC_E2E_00055, UC_E2E_00057, UC_E2E_00061, UC_E2E_00304, UC_E2E_00317, UC_E2E_00328</td>
<td></td>
</tr>
<tr>
<td>SRS_E2E_08529</td>
<td>Each of the defined E2E profiles shall use an appropriate subset of specific protection mechanisms</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SRS_E2E_08530</td>
<td>Each E2E profile shall have a unique ID, define precisely a set of mechanisms and its behavior in a semi-formal way</td>
</tr>
<tr>
<td>SRS_E2E_08531</td>
<td>E2E library shall call the CRC routines of CRC library</td>
</tr>
<tr>
<td>SRS_E2E_08533</td>
<td>CRC used in a E2E profile shall be different than the CRC used by the underlying physical communication protocol</td>
</tr>
<tr>
<td>SRS_E2E_08534</td>
<td>E2E library shall provide separate error flags and error counters for each type of detected communication failure</td>
</tr>
</tbody>
</table>
Either SW-C or E2E Library shall compute the intermediate CRC over application data element [Approved]

When using E2E Profiles 1/2, SW-Cs shall tolerate at least one received data element that is invalid/corrupted but not detected by E2E

An E2E protection mechanism for inter-ECU communication of large data shall be provided
The functional behavior of each library function shall not be configurable

<table>
<thead>
<tr>
<th>SRS_LIBS_00001</th>
<th>SWS_E2E_NA_00294</th>
</tr>
</thead>
<tbody>
<tr>
<td>The functional behavior of each library function shall not be configurable</td>
<td>SWS_E2E_00448, SWS_E2E_00449, SWS_E2E_00450, SWS_E2E_00451, SWS_E2E_00452, SWS_E2E_00453, SWS_E2E_00454, SWS_E2E_00455, SWS_E2E_00456, SWS_E2E_00457, SWS_E2E_00458, SWS_E2E_00459, SWS_E2E_00460, SWS_E2E_00461, SWS_E2E_00462, SWS_E2E_00466, SWS_E2E_00469, SWS_E2E_00470, SWS_E2E_00478, SWS_E2E_00481, SWS_E2E_00482, SWS_E2E_00483, SWS_E2E_00484, SWS_E2E_00486, SWS_E2E_00487, SWS_E2E_00490, SWS_E2E_00491, SWS_E2E_00492, SWS_E2E_00493, SWS_E2E_00494, SWS_E2E_00495, SWS_E2E_00496, SWS_E2E_00497, SWS_E2E_00498, SWS_E2E_00499, SWS_E2E_00500, SWS_E2E_00501, SWS_E2E_00509, SWS_E2E_00510, SWS_E2E_00511, SWS_E2E_00512, SWS_E2E_00513, SWS_E2E_00514, SWS_E2E_00515, SWS_E2E_00516, SWS_E2E_00517, SWS_E2E_00518, SWS_E2E_00519, SWS_E2E_00521, SWS_E2E_00523, SWS_E2E_00524, SWS_E2E_00525, SWS_E2E_00526, SWS_E2E_00527, SWS_E2E_00528, SWS_E2E_00529, SWS_E2E_00530, SWS_E2E_00531, SWS_E2E_00532, SWS_E2E_00533, SWS_E2E_00534, SWS_E2E_00535, SWS_E2E_00536, SWS_E2E_00537, SWS_E2E_00539, SWS_E2E_00542, SWS_E2E_00544, SWS_E2E_00545, SWS_E2E_00546, SWS_E2E_00547, SWS_E2E_00548, SWS_E2E_00549, SWS_E2E_00550, SWS_E2E_00551, SWS_E2E_00552, SWS_E2E_00553, SWS_E2E_00554, SWS_E2E_00555, SWS_E2E_00556, SWS_E2E_00557, SWS_E2E_00558, SWS_E2E_00559, SWS_E2E_00560, SWS_E2E_00561, SWS_E2E_00562, SWS_E2E_00563, SWS_E2E_00564, SWS_E2E_00565, SWS_E2E_00567, SWS_E2E_00572, SWS_E2E_00573, SWS_E2E_00574, SWS_E2E_00575, SWS_E2E_00576, SWS_E2E_00577, SWS_E2E_00578, SWS_E2E_00579, SWS_E2E_00580, SWS_E2E_00581, SWS_E2E_00582, UC_E2E_00236, UC_E2E_00316, UC_E2E_00327, UC_E2E_00463, UC_E2E_00464</td>
</tr>
<tr>
<td>Specification ID</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SRS_LIBS_00002</td>
<td>A library shall be operational before all BSW modules and application SW-Cs</td>
</tr>
<tr>
<td>SRS_LIBS_00003</td>
<td>A library shall be operational until the shutdown</td>
</tr>
<tr>
<td>SRS_LIBS_00004</td>
<td>Using libraries shall not pass through a port interface</td>
</tr>
<tr>
<td>SRS_LIBS_00005</td>
<td>Each library shall provide one header file with its public interface</td>
</tr>
<tr>
<td>SRS_LIBS_00007</td>
<td>Using a library should be documented</td>
</tr>
<tr>
<td>SRS_LIBS_00008</td>
<td>For a given function prototype name, the behavior and the parameters shall not evolve once it is a part of an AUTOSAR final release</td>
</tr>
<tr>
<td>SRS_LIBS_00009</td>
<td>All library functions shall be re-entrant</td>
</tr>
<tr>
<td>SRS_LIBS_00010</td>
<td>A library shall define its own specific types in the library header file if and only if they are not yet defined by AUTOSAR</td>
</tr>
<tr>
<td>SRS_LIBS_00011</td>
<td>All function names and type names shall start with &quot;Library short name_&quot;</td>
</tr>
<tr>
<td>SRS_LIBS_00012</td>
<td>Passing parameters with structure shall be allowed [approved]</td>
</tr>
<tr>
<td>SRS_LIBS_00013</td>
<td>The error cases, resulting in the check at runtime of the value of input parameters, shall be listed in SWS</td>
</tr>
<tr>
<td>SRS_LIBS_00015</td>
<td>It shall be possible to configure the microcontroller so that the library code is shared between all callers</td>
</tr>
<tr>
<td>SRS_LIBS_00016</td>
<td>A SW-C may use a non-AUTOSAR library available on the market</td>
</tr>
<tr>
<td>SRS_LIBS_00017</td>
<td>Usage of macros should be avoided</td>
</tr>
<tr>
<td>SRS_LIBS_00018</td>
<td>A library function may only call library functions</td>
</tr>
<tr>
<td>SRS_LIBS_08518</td>
<td>The CRC Library shall provide different calculation methods, optimizing either performance or memory usage</td>
</tr>
<tr>
<td>SRS_LIBS_08521</td>
<td>All CRC routines shall allow step-by-step-wise calculation of a large data block</td>
</tr>
<tr>
<td>SRS_LIBS_08525</td>
<td>The CRC library shall support the standard generator polynomials</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>SRS_LIBS_08526</td>
<td>The CRC Library shall support current standards of CRC calculation</td>
</tr>
</tbody>
</table>
7 Functional specification

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

7.1 Overview of communication protection

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

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

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

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

E2E protection works as follows:
- Sender: addition of control fields like CRC or counter to the transmitted data;
- Receiver: evaluation of the control fields from the received data, calculation of control fields (e.g. CRC calculation on the received data), comparison of calculated control fields with an expected/received content.

![Diagram of safety protocol concept](image)

Figure 7-1: Safety protocol concept (with exemplary location of the E2E header)
Each E2E profile has a specific set of control fields with a specific functional behavior and with specific properties for the detection of communication faults.

### 7.2 Overview of E2E Profiles

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

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

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

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

Depending on the used communication and network stack, appropriate subsets of these mechanisms are defined as E2E communication profiles.\(\text{[SRS_E2E_08531]}\)

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

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

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

**[SWS_E2E_00217]** The implementation of the E2E Library shall provide at least one of the E2E Profiles.\(\text{[SRS_E2E_08528]}\)

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

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

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

[SWS_E2E_00011] The E2E Library shall report errors detected by library-internal mechanisms to callers of E2E functions through return value.] (SRS_E2E_08528)

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


### Specification of SW-C End-to-End Communication Protection Library

AUTOSAR CP Release 4.3.1

---

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

<table>
<thead>
<tr>
<th>Type or error or status</th>
<th>How do caller of E2E shall handle it</th>
<th>Related code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least one pointer parameter is a NULL pointer</td>
<td>Development error or Integration error</td>
<td>E2E_E_INPUTERR_NULL</td>
<td>0x13</td>
</tr>
<tr>
<td>At least one input parameter is erroneous, e.g. out of range</td>
<td>Development error or Integration error</td>
<td>E2E_E_INPUTERR_WRONG</td>
<td>0x17</td>
</tr>
<tr>
<td>An internal library error has occurred (e.g. error detected by program flow monitoring, violated invariant or postcondition)</td>
<td>Development error or Integration error</td>
<td>E2E_E_INTERR</td>
<td>0x19</td>
</tr>
<tr>
<td>Function completed successfully</td>
<td>N/A</td>
<td>E2E_E_OK</td>
<td>0x00</td>
</tr>
<tr>
<td>Function executed in wrong state</td>
<td>Development error or integration error</td>
<td>E2E_E_WRONGSTATE</td>
<td>0x1A</td>
</tr>
</tbody>
</table>

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

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

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

[UC_E2E_00313] The caller of the E2E functions E2E_PXXProtect() / E2E_PXXCheckshall handle the errors/stati defined in SWS_E2E_00047 according to the column “How do caller of E2E shall handle it”. [SRS_E2E_08528]

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

#### 7.2.2 Error detection

[SWS_E2E_00012] The internal library mechanisms shall detect and report errors shall be implemented according to the pre-defined E2E Profiles specified in sections 7.3 and 7.4. [SRS_E2E_08528]

### 7.3 Specification of E2E Profile 1

Profile 1 shall provide the following mechanisms:
### Mechanism | Description
--- | ---
**Counter** | 4bit (explicitly sent) representing numbers from 0 to 14 incremented on every send request. Both Alive Counter and Sequence Counter mechanisms are provided by E2E Profile 1, evaluating the same 4 bits.

**Timeout monitoring** | Timeout is determined by E2E Library by means of evaluation of the Counter, by a non-blocking read at the receiver. Timeout is reported by E2E Library to the caller by means of the status flags in E2E_P01CheckStatusType.

**Data ID** | 16 bit, unique number, included in the CRC calculation. For dataIdMode equal to 0, 1 or 2, the Data ID is not transmitted, but included in the CRC computation (implicit transmission).

For dataIdMode equal to 3:
- the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,
- the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data,
- the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for dataIdMode equal to 0, 1 or 2).

**CRC** | CRC-8-SAE J1850 - 0x1D (x8 + x4 + x3 + x2 + 1), but with different start and XOR values (both start value and XOR value are 0x00).

This CRC is provided by CRC library. Starting with AUTOSAR R4.0, the SAE8 CRC function of the CRC library uses 0xFF as start value and XOR value. To compensate a different behavior of the CRC library, the E2E Library applies additional XOR 0xFF operations starting with R4.0, to come up with 0x00 as start value and XOR value.

Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN.

| (SRS_E2E_08529, SRS_E2E_08530, SRS_E2E_08533) |
--- |

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

<table>
<thead>
<tr>
<th>E2E Mechanism</th>
<th>Detected communication faults</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Counter</strong></td>
<td>Repetition, Loss, insertion, incorrect sequence, blocking</td>
</tr>
<tr>
<td>Transmission on a regular basis and timeout monitoring using E2E-Library(^1)</td>
<td>Loss, delay, blocking</td>
</tr>
<tr>
<td>Data ID + CRC</td>
<td>Masquerade and incorrect addressing, insertion</td>
</tr>
<tr>
<td>CRC (^2)</td>
<td>Corruption, Asymmetric information</td>
</tr>
</tbody>
</table>

\(^1\) Implementation by sender and receiver, which are using E2E-Library

\(^2\) for a set of data protected by same CRC
[SWS_E2E_00070] E2E Profile 1 shall use the polynomial of CRC-8-SAE J1850, i.e. the polynomial \(0x1D (x^8 + x^4 + x^3 + x^2 + 1)\), but with start value and XOR value equal to \(0x00\). (SRS_E2E_08531)

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

### 7.3.1 Data Layout

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

### 7.3.2 Counter

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

[SWS_E2E_00075] In E2E Profile 1, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request (from sender SW-C). When the counter reaches the value 14 (0xE), then it shall restart with 0 for the next send request (i.e. value 0xF shall be skipped). All these actions shall be executed by E2E Library.] (SRS_E2E_08528)

[SWS_E2E_00076] In E2E Profile 1, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following shall be detected: (1) no new data has arrived since last invocation of E2E library check function, (2) no new data has arrived since receiver start, (3) the data is repeated (4) counter is incremented by one (i.e. no data lost), (5) counter is incremented more than by one, but still within allowed limits (i.e. some data lost), (6) counter is incremented more than allowed (i.e. too many data lost). All these actions shall be executed by E2E Library.] (SRS_E2E_08528)

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

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

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

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

1. E2E_P01_DATAID_BOTH: both two bytes (double ID configuration) are included in the CRC, first low byte and then high byte (see variant 1A - SWS_E2E_00227) or
2. E2E_P01_DATAID_ALT: depending on parity of the counter (alternating ID configuration) the high and the low byte is included (see variant 1B - SWS_E2E_00228). For even counter values the low byte is included and for odd counter values the high byte is included.
3. E2E_P01_DATAID_LOW: only the low byte is included and high byte is never used. This equals to the situation if the Data IDs (in a given application) are only 8 bits.
4. E2E_P01_DATAID_NIBBLE:
   - the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,
   - the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.
   - the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for the inclusion modes BOTH, ALT and LOW)] (SRS_E2E_08528)

[SWS_E2E_00085] In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_BOTH or E2E_P01_DATAID_ALT the length of the Data ID shall be 16 bits (i.e. 2 byte).] (SRS_E2E_08528)

[SWS_E2E_00169] In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_LOW, the high byte of Data ID shall be set to 0x00.] (SRS_E2E_08528)

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

[SWS_E2E_00306] In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_NIBBLE, the high nibble of the high byte shall be 0x0.] (SRS_E2E_08528)

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

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

### 7.3.4 CRC calculation

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

**[SWS_E2E_00083]** E2E Profile 1 shall use CRC-8-SAE J1850 for CRC calculation. It shall use 0x00 as the start value and XOR value. (SRS_E2E_08529, SRS_E2E_08533)

**[SWS_E2E_00190]** E2E Profile 1 shall use the Crc_CalculateCRC8() function of the SWS CRC Library for calculating CRC checksums. (SRS_E2E_08528, SRS_E2E_08531)

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

The CRC calculation is illustrated by the following two examples.

For standard variant 1A:

![CRC calculation example](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIgAAAAH/CAAAACmrPvAAAABGdBTUEAALGPC/xhBQAAAAFzSFr0lMMAAAAXR0lMAAASIAADhMMAAAABmJLR0hTDAFAAAWHOOFWELCwAAAAhJREFUeNrs7P/tP/sAAAAABJRU5ErkJggg==)

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

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

For standard variant 1C:
The Data ID can be encoded in CRC in different ways, see SWS_E2E_00163.

[SWS_E2E_00082] In E2E Profile 1, the CRC is calculated over:
1. First over the one or two bytes of the Data ID (depending on Data ID configuration) and then
Over all transmitted bytes of a safety-related complex data element/signal group (except the CRC byte).] (SRS_E2E_08536)

7.3.5 Timeout detection

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

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

7.3.6 E2E Profile 1 variants

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

[SWS_E2E_00227] The E2E Profile variant 1A is defined as follows:
1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_BOTH
4. SignallPdu.unusedBitPattern = 0xFF.} (SRS_E2E_08528)

[SWS_E2E_00228] The E2E Profile variant 1B is defined as follows:
1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_ALTERNATING
4. SignallPdu.unusedBitPattern = 0xFF. (SRS_E2E_08528)

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

<table>
<thead>
<tr>
<th>CRC</th>
<th>Sig0</th>
<th>Alive</th>
<th>Sig1</th>
<th>Sig2</th>
<th>CRC</th>
<th>1111</th>
<th>Alive</th>
<th>Sig1[3]</th>
</tr>
</thead>
</table>

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

[SWS_E2E_00307] The E2E Profile variant 1C is defined as follows:
1. CRC is the 0\textsuperscript{th} byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1\textsuperscript{st} byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1\textsuperscript{st} byte (i.e. starts with bit offset 12)
4. E2E_P01DataIDMode = E2E_P01_DATAID_NIBBLE
5. SignallPdu.unusedBitPattern = 0xFF. (SRS_E2E_08528)

7.3.7 E2E_P01Protect

[SWS_E2E_00195] The function E2E_P01Protect() shall:
1. write the Counter in Data,
2. write DataID nibble in Data (E2E_P01_DATAID_NIBBLE) in Data
3. compute the CRC over DataID and Data
4. write CRC in Data
5. increment the Counter (which will be used in the next invocation of E2E_P01Protect()),
as specified by Figure 7-5 and Figure 7-6. (SRS_E2E_08528)
**Figure 7-5: E2E_P01Protect()**

### 7.3.8 Calculate CRC

The diagram of the function E2E_P01Protect() (see above chapter) and E2E_P01Check() (see below chapter) have a sub-diagram specifying the calculation of CRC:
**Figure 7-6: Subdiagram “Calculate CRC over Data ID and Data”, used by E2E_P01Protect() and E2E_P01Check()**

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

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

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

To have the same results of the functions E2E_P01Protect() and E2E_P02Check() in >=R4.0 and R3.2, while using differently functioning CRC library, E2E “compensates” different behavior of the CRC library. This results with different invocation of the CRC library by E2E library (see Figure 7-6) in >=R4.0 and R3.2. This means Figure 7-6 is different in >=R4.0 and R3.2.
7.3.9 E2E_P01Check

[SWS_E2E_00196] The function E2E_P01Check shall
1. Check the CRC
2. Check the Data ID nibble, i.e. compare the expected value with the received value (for E2E_P01_DATAID_NIBBLE configuration only)
3. Check the Counter,
4. determine the check Status,

as specified by Figure 7-7 and Figure 7-6.\] (SRS_E2E_08528, SRS_E2E_08530)
Figure 7-7: E2E_P01Check()

The diagram of the function E2E_P01Check() has a sub-diagram specifying the calculation of CRC, which is shown by Figure 7-6.
7.4 Specification of E2E Profile 2

Profile 2 shall provide the following mechanisms:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number (Counter)</td>
<td>4-bit (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.</td>
</tr>
<tr>
<td>Message Key used for CRC calculation (Data ID)</td>
<td>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.</td>
</tr>
<tr>
<td>Safety Code (CRC)</td>
<td>8-bit explicitly sent (Data[0]) Polynomial: 0x2F (x^8 + x^5 + x^3 + x^2 + x + 1) Start value: 0xFF Final XOR-value: 0xFF Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay and CAN.</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>E2E Mechanism</th>
<th>Detected communication faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>Repetition, Loss, insertion, incorrect sequence, blocking</td>
</tr>
<tr>
<td>Transmission on a regular bases and timeout monitoring using E2E-Library(^1)</td>
<td>Loss, delay, blocking</td>
</tr>
<tr>
<td>Data ID + CRC</td>
<td>Masquerade and incorrect addressing, insertion</td>
</tr>
<tr>
<td>CRC</td>
<td>Corruption, Asymmetric information(^2)</td>
</tr>
</tbody>
</table>

\(^1\) Implementation by sender and receiver for a set of data protected by same CRC

Table 7-2: Detectable communication faults using Profile 2
[SWS_E2E_00117] E2E Profile 2 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Library for calculating CRC checksums. (SRS_E2E_08531)

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

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

[SWS_E2E_00120] In E2E Profile 2, the specific Data ID used for CRC calculation shall be selected from a pre-defined DataIDList[16] using the value of the Counter as an index. (SRS_E2E_08528)

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

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

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

An example DataIDList is presented in Chapter 13.4.

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

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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>Counter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

] (SRS_E2E_08528)

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

[SWS_E2E_00123] In E2E Profile 2, the Counter shall be the low nibble (Bit 0…Bit 3) of Data[1]. (SRS_E2E_08528)
[SWS_E2E_00124] In E2E Profile 2, the E2E_P02Protect() function shall not modify any bit of Data except the bits representing the CRC and the Counter.] (SRS_E2E_08528)

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

7.4.1 E2E_P02Protect

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

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

![Flow chart of E2E_P02Protect()](image)

**Figure 7-8: E2E_P02Protect()**
Figure 7-9: Increment Counter

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

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

[SWS_E2E_00129] When the Counter has reached its upper bound of 15 (0xF), it shall restart at 0 for the next call of the E2E_P02Protect() from the sending SW-C.]} (SRS_E2E_08528)

[SWS_E2E_00130] In E2E Profile 2, the E2E_P02Protect() function shall update the Counter (i.e. low nibble (Bit 0…Bit 3) of Data byte 1) in the data buffer (Data) after incrementing the Counter.] (SRS_E2E_08528)

The specific Data ID used for this send request is then determined from a DataIDList[] depending on the value of the Counter (Counter is used as an index to select the Data ID from DataIDList[]). The DataIDList[] is defined in E2E_P02ConfigType.
[SWS_E2E_00132] In E2E Profile 2, after determining the specific Data ID, the E2E_P02Protect() function shall calculate the CRC over Data[1], Data[2], … Data[Config->DataLength/8-1] of the data buffer (Data) extended with the Data ID.
(SRS_E2E_08528)

[SWS_E2E_00133] In E2E Profile 2, the E2E_P02Protect() function shall update the CRC (i.e. Data[0]) in the data buffer (Data) after computing the CRC.
(SRS_E2E_08528)

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

**7.4.2 E2E_P02Check**

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

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

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

CalculatedCRC = Crc_CalculateCRC8H2F() computed over Data[1], Data[2], … Data[Config->DataLength/8-1], DataID

CalculatedCRC == Data[0]

return TRUE

return FALSE

Figure 7-11: Checksum OK
Calculate DeltaCounter

\[
\text{DeltaCounter = ReceivedCounter - State->LastValidCounter}
\]

\(\text{(sint8) DeltaCounter < 0?}\)

\[
\begin{align*}
\text{[TRUE]} \\
\text{DeltaCounter = DeltaCounter + 16} \\
\text{[FALSE]} \\
\end{align*}
\]

return

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:
  
  \[
  \text{CalculatedCRC} = \text{Crc_CalculateCRC8H2F()} \text{ calculated over Data[1], Data[2], ... Data[Config->DataLength/8-1], Data ID}
  \]
Finally, the check for correctness of the received Data is performed by comparing CalculatedCRC with the value of CRC stored in Data.

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

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

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

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

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

[SWS_E2E_00135] In E2E Profile 2, the local variable DeltaCounter shall be calculated by subtracting LastValidCounter from Counter in Data, considering an overflow due to the range of values [0…15].] (SRS_E2E_08528)

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

[SWS_E2E_00136] In E2E Profile 2, MaxDeltaCounter shall specify the maximum allowed difference between two Counter values of two consecutively received valid messages.] (SRS_E2E_08528)

[SWS_E2E_00137] In E2E Profile 2, MaxDeltaCounter shall be incremented by 1 every time the E2E_P02Check() function is called, up to the maximum value of 15 (0xF).] (SRS_E2E_08528)

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

[SWS_E2E_00139] In E2E Profile 2, the E2E_P02Check() function shall determine the specific Data ID from DataIDList using the Counter of the received Data as index. ] (SRS_E2E_08528)
Specification of SW-C End-to-End Communication Protection Library
AUTOSAR CP Release 4.3.1

[SWS_E2E_00140]: In E2E Profile 2, the E2E_P02Check() function shall calculate CalculatedCRC over Data[1], Data[2], … Data[Config->DataLength/8-1] of the data buffer (Data) extended with the determined Data ID. (SRS_E2E_08528)

[SWS_E2E_00141]: In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_WRONGCRC if the calculated CalculatedCRC value differs from the value of the CRC in Data. (SRS_E2E_08528)

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

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

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

[SWS_E2E_00145]: The E2E_P02Check() function shall
- set Status to E2E_P02STATUS_WRONGSEQUENCE; and
- re-initialize SyncCounter with SyncCounterInit if the calculated value of DeltaCounter exceeds the value of MaxDeltaCounter. (SRS_E2E_08528)

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

[SWS_E2E_00147]: The E2E_P02Check() function shall set Status to E2E_P02STATUS_OK if the following conditions are true:
- the calculated DeltaCounter equals 1; and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State -> NoNewOrRepeatedDataCounter <= Config -> MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0. (SRS_E2E_08528)

[SWS_E2E_00298]: The E2E_P02Check() function shall
- re-initialize SyncCounter with SyncCounterInit; and
- set Status to E2E_P02STATUS_SYNC; if the following conditions are true:
- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. 1 <= DeltaCounter <= MaxDeltaCounter); and
- AUTOSAR confidential -
[SWS_E2E_00300] The E2E_P02Check() function shall reset the NoNewOrRepeatedDataCounter to 0 if it returns one of the following status:
- E2E_P02STATUS_OK;
- E2E_P02STATUS_OKSOMELOST;
- E2E_P02STATUS_SYNC;
- E2E_P02STATUS_WRONGSEQUENCE] (SRS_E2E_08528)

[SWS_E2E_00301] The E2E_P02Check() function shall increment NoNewOrRepeatedDataCounter by 1 if it returns the Status E2E_P02STATUS_NONEWDATA or E2E_P02STATUS_REPEATED up to the maximum value of Counter (i.e. 15 or 0xF).] (SRS_E2E_08528)

7.5 Specification of E2E Profile 4

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

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

] (SRS_E2E_08529, SRS_E2E_08530, SRS_E2E_08533)

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

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

Table 7-3: Detectable communication faults using Profile 4
For details of CRC computation, the usage of start values and XOR values see CRC Library [7].

7.5.1 Data Layout

7.5.1.1 User data layout

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

7.5.1.2 Header layout

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

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

Figure 7-13: E2E Profile 4 header

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

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

For example, the 16 bits of the E2E counter are transmitted in the following order (higher number meaning higher significance):

7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7.

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

7.5.2 Counter

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

[SWS_E2E_00478] In E2E Profile 4, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF'FF), then it shall restart with 0 for the next send request.

(SRS_E2E_08539)

Note: This specification was previously falsely identified as SWS_E2E_00324.

Note that the counter value 0xFF'FF is not reserved as a special invalid value, but it is used as a normal counter value.
In E2E Profile 4, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

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

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

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

7.5.3 Data ID

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

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

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

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

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

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

7.5.4 Length

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

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

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

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

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

7.5.6 Timeout detection

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

7.5.7 E2E Profile 4 variants

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

7.5.8 E2E_P04Protect

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

[SWS_E2E_00362] The function E2E_P04Protect() shall have the following overall behavior:
The step “Verify inputs of the protect function” in E2E_P04Protect() shall have the following behavior:

```
E2E_P04Protect(Config, State, Data, Length)
  Write DataID
  Write Counter
  Compute CRC
  Increment Counter
  return E2E_E_OK
Write Length
Write Counter
Write DataID
Compute CRC
Write CRC
Increment Counter
  return E2E_E_INPUTERR_NULL
  [wrong input]
  [input ok]
return E2E_E_INPUTERR_WRONG
```

[SRS_E2E_08539]
The step “Compute offset” in E2E_P04Protect() and E2E_P04Check() shall have the following behavior:

```
Offset = Config->Offset / 8
```

The step “Write Length” in E2E_P04Protect() shall have the following behavior:

```
compute local variable uint16 Offset, which is in [byte]
```
The step “Write Counter” in E2E_P04Protect() shall have the following behavior:

The step “Write DataID” in E2E_P04Protect() shall have the following behavior:
The step “ComputeCRC” in E2E_P04Protect() and in E2E_P04Check() shall have the following behavior:

```
uint32 ComputedCRC = Crc_CalculateCRC32P4(Crc_DataPtr: &Data[0], Crc_Length: Offset+8, Crc_StartValue32: 0xFFFFFFFF, Crc_IsFirstCall: TRUE)
```

compute CRC over bytes that are before CRC.
computed length: offset+8, where:
- offset: number of bytes before the E2E header
- 8: number of header bytes before E2E CRC

At this step, there is a ready ComputedCRC value, over the entire E2E-Produced Data (including E2E Header [length, ID, CRC etc] and the user data).

The step “Write CRC” in E2E_P04Protect() shall have the following behavior:

```
uint32 ComputedCRC = Crc_CalculateCRC32P4(Crc_DataPtr: &Data[Offset+12], Crc_Length: Length-Offset-12, Crc_StartValue32: ComputedCRC, Crc_IsFirstCall: FALSE)
```

Compute CRC over bytes that are after CRC (if any).
The step “Increment Counter” in E2E_P04Protect() shall have the following behavior:

The type is uint16. After 0xFFFF, the next value is 0.

7.5.9 E2E_P04Check

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

[SWS_E2E_00355] The function E2E_P04Check() shall have the following overall behavior:
The step “Verify inputs of the check function” in `E2E_P04Check()` shall have the following behavior:

```
E2E_P04Check(Config, State, Data, Length)
Read DataID
Read Length
Read Counter
return E2E_E_OK
Read CRC
Verify inputs of the check function
Do checks
return E2E_E_INPUTERR_NULL
```

(SRS_E2E_08539)

[SWS_E2E_00356] The step “Verify inputs of the check function” in `E2E_P04Check()` shall have the following behavior:
The step “Read Length” in E2E_P04Check() shall have the following behavior:

Copy bytes Data[Offset...Offset+1] in Big Endian order to uint16 local variable ReceivedLength.

Options is always at the same location.
[SWS_E2E_00358] The step “Read Counter” in E2E_P04Check() shall have the following behavior:

Copy bytes Data[Offset+2...Offset+3] in BigEndian order on uint16 local variable ReceivedCounter

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

Copy bytes Data[Offset+4...Offset+7] in BigEndian order on uint32 local variable ReceivedDataID

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

1. Copy bytes Data[Offset+8...Offset+11] using big Endian order on 4-byte local variable ReceivedCRC.
2. ReceivedCRC == ComputedCRC
3. ReceivedDataID == Config->DataID
4. ReceivedLength == Length
5. (DeltaCounter <= Config->MaxDeltaCounter) && (DeltaCounter >=0)
6. State->Status = E2E_P04STATUS_ERROR
7. State->Status = E2E_P04STATUS_REPEATED
8. DeltaCounter > 0
9. State->Status = E2E_P04STATUS_OK
10. NewDataAvailable == TRUE
11. State->Status = E2E_P04STATUS_NONEWDATA
12. DeltaCounter == 1
13. State->Status = E2E_P04STATUS_OKSOMELOST
14. State->Status = E2E_P04STATUS_WRONGSEQUENCE
15. State->Counter = ReceivedCounter
16. Compute local variable DeltaCounter: ReceivedCounter - State->Counter (taking into wrap around 0xFFFF)
7.6 Specification of E2E Profile 5

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

<table>
<thead>
<tr>
<th>Control field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>8-bits. (explicitly sent)</td>
</tr>
<tr>
<td>CRC</td>
<td>16 bits, polynomial in normal form 0x1021 (Autosar notation), provided by CRC library. (explicitly sent)</td>
</tr>
<tr>
<td>Data ID</td>
<td>16-bits, unique system-wide. (implicitly sent)</td>
</tr>
</tbody>
</table>

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

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

Table 7-4: Detectable communication faults using Profile 5

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

7.6.1 Data Layout

7.6.1.1 User data layout

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

7.6.1.2 Header layout

The header of the E2E Profile 5 has one fixed layout, as follows:
The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

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

7.6.2 Counter

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

[SWS_E2E_00397] In E2E Profile 5, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF), then it shall restart with 0 for the next send request.

(SRS_E2E_08539)

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

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

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

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

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

7.6.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.
[SWS_E2E_00399] In the E2E Profile 5, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation.]
(SRS_E2E_08539)

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

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

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

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

7.6.4 Length

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

7.6.5 CRC

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

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

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

7.6.6 Timeout detection

The previously mentioned mechanisms (for Profile 5: CRC, Counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts. The independent execution of the receiver is required by E2EUSE0089.
The attribute State->NewDataAvailable == FALSE means that the transmission medium (e.g. RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P05STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

### 7.6.7 E2E_P05Protect

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

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

![Diagram of E2E_P05Protect](image)
[SWS_E2E_00404] The step “Verify inputs of the protect function” in E2E_P05Protect() shall have the following behavior:

- If (Config != NULL) && (State != NULL) && (Data != NULL), then:
  - input ok
  - Length == Config->DataLength/8

- If no input, then:
  - [FALSE]

- If wrong input, then:
  - [TRUE]

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

- E2E_P05Protect():
  - compute local variable uint16 Offset, which is in [byte]
  - Offset = Config->Offset / 8

(SRS_E2E_08539)

[SWS_E2E_00405] The step “Write Counter” in E2E_P05Protect() shall have the following behavior:
The step “Compute CRC” in E2E_P05Protect() and in E2E_P05Check shall have the following behavior:

Copy 1-byte State->Counter on byte Data[Offset+2] in Little Endian order

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

[SRS_E2E_08539]

[SWS_E2E_00407]
The step “Increment Counter” in E2E_P05Protect() shall have
the following behavior:

1. Copy 2-byte local variable CRC on bytes Data[Offset+0...Offset+1] using Little
   Endian order
2. State->Counter++
   - The type is uint8. After 0xFF, the next value is 0.

---

[SRS_E2E_00409]
7.6.8 E2E_P05Check

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

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

[SWS_E2E_00412] The step “Verify inputs of the check function” in E2E_P05Check() shall have the following behavior:
The step “Read Counter” in `E2E_P05Check()` shall have the following behavior:

**null input**

- NewDataAvailable = FALSE
  - (Config != NULL) && (State != NULL)
  - (Data != NULL && Length != 0) || (Data == NULL && Length == 0)
    - Data != NULL
      - Length == Config->DataLength/8
        - [TRUE]
        - [FALSE]
    - [FALSE]
  - [FALSE]
  - [TRUE]
  - NewDataAvailable = TRUE

**input ok**

- NewDataAvailable = FALSE
  - (Config != NULL) && (State != NULL)
  - [TRUE]
  - [FALSE]

**wrong input**

- NewDataAvailable = FALSE
  - (Config != NULL) && (State != NULL)
  - [TRUE]
  - [FALSE]

---

[SRS_E2E_08539]

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

- Copy byte Data[Offset+2] in Little Endian order on uint8 local variable ReceivedCounter
- Copy bytes Data[Offset+0...Offset+1] using Little Endian order on 2-byte local variable ReceivedCRC

[SWS_E2E_00414]
The step “Do Checks” in E2E_P05Check() shall have the following behavior:

### Specification of E2E Profile 6

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

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

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

<table>
<thead>
<tr>
<th>Fault</th>
<th>Main safety mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repetition of information</td>
<td>Counter</td>
</tr>
<tr>
<td>Loss of information</td>
<td>Counter</td>
</tr>
<tr>
<td>Delay of information</td>
<td>Counter</td>
</tr>
<tr>
<td>Insertion of information</td>
<td>Data ID</td>
</tr>
<tr>
<td>Masquerading</td>
<td>Data ID, CRC</td>
</tr>
</tbody>
</table>
### Incorrect addressing
- Data ID

### Incorrect sequence of information
- Counter

### Corruption of information
- CRC

### Asymmetric information sent from a sender to multiple receivers
- CRC (to detect corruption at any of receivers)

### Information from a sender received by only a subset of the receivers
- Counter (loss on specific receivers)

### Blocking access to a communication channel
- Counter (loss or timeout)

| Table 7-5: Detectable communication faults using Profile 6 |

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

#### 7.7.1 Data Layout

#### 7.7.1.1 User data layout

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

#### 7.7.1.2 Header layout

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

<table>
<thead>
<tr>
<th>Transmission order</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>E2E Counter</td>
<td>E2E CRC</td>
<td>E2E Length</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>E2E CRC</td>
<td>E2E Length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7-15: E2E Profile 6 header**

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

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

#### 7.7.2 Counter

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

[SWS_E2E_00417] In E2E Profile 6, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the
maximum value (0xFF), then it shall restart with 0 for the next send request.] (SRS_E2E_08539)

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

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

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

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

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

7.7.3 Data ID

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

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

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

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

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

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

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

7.7.5 CRC

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

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

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

7.7.6 Timeout detection

The previously mentioned mechanisms (for Profile 6: CRC, Counter, Data ID, Length) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts. The independent execution of the receiver is required by E2EUSE0089.

The attribute State->NewDataAvailable == FALSE means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P06STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

7.7.7 E2E_P06Protect

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

[SWS_E2E_00423] The function E2E_P06Protect() shall have the following overall behavior:
The step “Verify inputs of the protect function” in E2E_P06Protect() shall have the following behavior:
The step “Compute offset” in E2E_P06Protect() and E2E_P06Check() shall have the following behavior:

\[
\text{Offset} = \frac{\text{Config} \rightarrow \text{Offset}}{8}
\]

The step “Write Length” in E2E_P06Protect() shall have the following behavior:
The step “Write Counter” in E2E_P06Protect() shall have the following behavior:

Copy 2-byte Length on bytes Data[Offset+2...Offset+3] in Big Endian order

[SWS_E2E_00426] The step “Compute CRC” in E2E_P06Protect() and E2E_P06Check() shall have the following behavior:

Copy 1-byte State->Counter on byte Data[Offset+4] in Big Endian order

[SWS_E2E_00427]
The step “Write CRC” in E2E_P06Protect() shall have the following behavior:
The step “Increment Counter” in E2E_P06Protect() shall have the following behavior:

Copy 2-byte local variable CRC on bytes Data[Offset+0...Offset+1] using big Endian order

The type is uint8. After 0xFF, the next value is 0.

[SWS_E2E_00429]: The step “Increment Counter” in E2E_P06Protect() shall have the following behavior:
7.7.8 E2E_P06Check

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

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

[SWS_E2E_00431] The step “Verify inputs of the check function” in E2E_P06Check() shall have the following behavior:
The step “Read Length” in E2E_P06Check() shall have the following behavior:

[SRS_E2E_08539] This path may happen at runtime if queued communication is used and no data is available (in this case both Data is NULL and Length is 0).

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

Copy bytes Data[Offset+2...Offset+3] in Big Endian order to uint16 local variable ReceivedLength

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

Copy byte Data[Offset+4] in Big Endian order on uint8 local variable ReceivedCounter
The step “Do Checks” in E2E_P06Check() shall have the following behavior:

Copy bytes Data[Offset+0...Offset+1] using big Endian order on 2-byte local variable ReceivedCRC

[SWS_E2E_00436] The step “Do Checks” in E2E_P06Check() shall have the following behavior:
7.8 Specification of E2E Profile 7

Profile 7 shall provide the following control fields, transmitted at runtime together with the protected data:

<table>
<thead>
<tr>
<th>Control field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>32 bits, to support dynamic-size data.</td>
</tr>
<tr>
<td>Counter</td>
<td>32-bits.</td>
</tr>
<tr>
<td>CRC</td>
<td>64 bits, polynomial in normal form 0x42F0E1EBA9EA3693, provided by CRC library.</td>
</tr>
</tbody>
</table>
|               | Note: This CRC polynomial is also known as “CRC-64 (ECMA)”.
| Data ID       | 32-bits, unique system-wide. |

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

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

Table 7-6: Detectable communication faults using Profile 7

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

### 7.8.1 Data Layout

#### 7.8.1.1 User data layout

In the E2E Profile 7, the user data layout (of the data to be protected) is not constrained by E2E Profile 7 – there is only a requirement that the length of data to be protected is multiple of 1 byte.

#### 7.8.1.2 Header layout

The header of the E2E Profile 7 has one fixed layout, as follows:

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

Figure 7-16: E2E Profile 7 header

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

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

For example, the 32 bits of the E2E counter are transmitted in the following order (higher number meaning higher significance):
24 25 26 27 28 29 30 31 16 17 18 19 20 21 22 23 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7.

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

In E2E Profile 7, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E library.

[SWS_E2E_00481] In E2E Profile 7, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF'FF'FF'FF), then it shall restart with 0 for the next send request.

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

In E2E Profile 7, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

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

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

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

7.8.3 Data ID

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

[SWS_E2E_00482] In the E2E Profile 7, the Data ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.

[SWS_E2E_00483] In the E2E profile 7, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data).

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

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

### 7.8.4 Length

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

### 7.8.5 CRC

E2E Profile 7 uses a 64-bit CRC, to ensure a high detection rate and high Hamming Distance.

[SWS_E2E_00484] E2E Profile 7 shall use the Crc_CalculateCRC64() function of the SWS CRC Library for calculating the CRC. [SRS_E2E_08539, SRS_E2E_08531]

[SWS_E2E_00485] In E2E Profile 7, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes) and over the user data. [SRS_E2E_08536]

### 7.8.6 Timeout detection

The previously mentioned mechanisms (CRC, Counter, Data ID, Length) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e., when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e., checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts. The independent execution of the receiver is required by UC_E2E_00089.

### 7.8.7 E2E Profile 7 variants

The E2E Profile 7 variants are specified in TPS System Specification.

### 7.8.8 E2E_P07Protect

The function E2E_P07Protect() performs the steps as specified by the following eight diagrams in this section.

[SWS_E2E_00486] The function E2E_P07Protect() shall have the following overall behavior:
The step “Verify inputs of the protect function” in E2E_P07Protect() shall have the following behavior:
**[SRS_E2E_08539]**

**[SWS_E2E_00488]** The step “Compute offset” in `E2E_P07Protect()` and `E2E_P07Check()` shall have the following behavior:

- `Offset = Config->Offset / 8`
- **compute local variable uint32 Offset, which is in [byte]**

**[SRS_E2E_08539]**

**[SWS_E2E_00489]** The step “Write Length” in `E2E_P07Protect()` shall have the following behavior:
[SWS_E2E_00490] The step “Write Counter” in E2E_P07Protect() shall have the following behavior:

Copy 4-byte State->Counter on bytes Data[Offset+12...Offset+15] in Big Endian order

[SWS_E2E_00491] The step “Write DataID” in E2E_P07Protect() shall have the following behavior:
The step “Compute CRC” in E2E_P07Protect() and in E2E_P07Check() shall have the following behavior:

- **E2E_P07Protect()**
- **E2E_P07Check()**

The step “Write CRC” in E2E_P07Protect() shall have the following behavior:

- **[SWS_E2E_00492]** The step “Compute CRC” in E2E_P07Protect() and in E2E_P07Check() shall have the following behavior:

```c
uint64 ComputedCRC = Crc_CalculateCRC64(Crc_DataPtr: &Data[offset+8], Crc_Length: Length-Offset-8, Crc_StartValue64: 0xFFFFFFFFFFFFFFFF, Crc_IsFirstCall: FALSE)
```

- **[SWS_E2E_00493]** The step “Write CRC” in E2E_P07Protect() shall have the following behavior:

```c
uint64 ComputedCRC = Crc_CalculateCRC64(Crc_DataPtr: &Data[0], Crc_Length: Offset, Crc_StartValue64: 0xFFFFFFFFFFFFFFFF, Crc_IsFirstCall: TRUE)
```
[SRS_E2E_08539]

The step “Increment Counter” in E2E_P07Protect() shall have the following behavior:

The type is uint32. After 0xFFFFFFFF, the next value is 0.

[SWS_E2E_00494] 7.8.9 E2E_P07Check

The function E2E_P07Check performs the actions as specified by the following seven diagrams in this section and according to diagram SWS_E2E_00492.

[SWS_E2E_00495] The function E2E_P07Check() shall have the following overall behavior:
The step “Verify inputs of the check function” in E2E_P07Check() shall have the following behavior:
The step “Read Length” in E2E_P07Check() shall have the following behavior:

Copy bytes Data[Offset+8...Offset+11] in Big Endian order to uint32 local variable ReceivedLength
[SWS_E2E_00498] The step “Read Counter” in E2E_P07Check() shall have the following behavior:

```c
void E2E_P07Check()
{
    // Copy bytes Data[Offset+12...Offset+15] in Big Endian order on uint32 local variable ReceivedCounter
}
```

[SWS_E2E_00499] The step “Read DataID” in E2E_P07Check() shall have the following behavior:

```c
void E2E_P07Check()
{
    // Copy bytes Data[Offset+16...Offset+19] in Big Endian order on uint32 local variable ReceivedDataID
}
```

[SWS_E2E_00500] The step “Read CRC” in E2E_P07Check() shall have the following behavior:
The step “Do Checks” in E2E_P07Check() shall have the following behavior:

- ReceivedCRC == ComputedCRC
- ReceivedDataID == Config->DataID
- ReceivedLength == Length
- (DeltaCounter <= Config->MaxDeltaCounter) && (DeltaCounter >= 0)
- State->Status = E2E_P07STATUS_ERROR
- State->Status = E2E_P07STATUS_REPEATED
- State->Status = E2E_P07STATUS_OK
- State->Status = E2E_P07STATUS_WRONGSEQUENCE
- State->Status = E2E_P07STATUS_NONEWDATA
- State->Status = E2E_P07STATUS_OKSOMELOST

Compute local variable DeltaCounter: ReceivedCounter - State->Counter (taking into wrap around 0xFF'FF'FF'FF)

[SWS_E2E_00501]
7.9 Specification of E2E Profile 11

Profile 11 is bus-compatible to profile 1, but provides „new“ profile behavior similar to profiles 4 to 7 on receiver side. Moreover, some legacy DataIDModes that are by now obsolete are omitted.

Profile 11 shall provide the following control fields, transmitted at runtime together with the protected data:

<table>
<thead>
<tr>
<th>Control field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>4-bits. (explicitly sent)</td>
</tr>
<tr>
<td>CRC</td>
<td>8 bits, CRC-8-SAE J1850, provided by CRC library. (explicitly sent)</td>
</tr>
<tr>
<td>Data ID</td>
<td>16-bits or 12-bits, unique system-wide. (either implicitly sent (16-bits) or partly explicitly sent (12-bits; 4 bits explicitly and 8 bits implicitly sent))</td>
</tr>
</tbody>
</table>

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

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

Table 7-7: Detectable communication faults using Profile 11

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

7.9.1 Data Layout

7.9.1.1 User data layout

In the E2E Profile 11, the user data layout (of the data to be protected) is not constrained by E2E Profile 11 – there is only a requirement, that the length of data to be protected is multiple of 1 byte.
7.9.1.2 Header layout

Profile 11 is backward compatible to the bus-layout of profile 1. This means that while all the header fields are configurable, the profile variants of profile 1 are also applicable. Namely, profile 1 variant 1A and variant 1C (see also SWS_E2E_00227 and SWS_E2E_00307).

<table>
<thead>
<tr>
<th>Byte Order</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission Order</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Bit Order</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 7-17: E2E Profile11 header according to Profile 11 variant 11C.

The figure above shows Profile 11 variant 11C where the configuration is given as:
- CRCOffset = 0
- CounterOffset = 8
- DataIDNibbleOffset = 12

For Profile 11 Variant 11A, DataIDNibble is not used. Instead, user data can be placed there.

[SWS_E2E_00540]: The E2E Profile variant 11A is defined as follows:
1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P11DataIDMode = E2E_P11_DATAID_BOTH
4. SignallPdu.unusedBitPattern = 0xFF.

Below is an example compliant to 11A:

<table>
<thead>
<tr>
<th>Byte Order</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission Order</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Bit Order</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 7-18: E2E Profile 11 variant 11A

[SWS_E2E_00541]: The E2E Profile variant 11C is defined as follows:
1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1st byte (i.e. starts with bit offset 12)
4. E2E_P11DataIDMode = E2E_P11_DATAID_NIBBLE
5. SignallPdu.unusedBitPattern = 0xFF.

E2E Profile variant 11C relates to Configuration of E2E Profile 11 configuration setting 11C in system template (system template is more specific).
The transmission order shown above represents the order in which bits are transmitted. For comparability to the figures of profile 1, also the bit order is given. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

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

### 7.9.2 Counter

In E2E Profile 11, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E library.

[SWS_E2E_00504] | In E2E Profile 11, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0x0E), then it shall restart with 0 for the next send request.] (SRS_E2E_08539)

Note that the counter value 0xF is reserved as a special invalid value, and must never be used by the E2E profile 11.

In E2E Profile 11, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

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

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

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

### 7.9.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element. There are two supported modes how the Data ID is used:
1. **E2E_P11_DATAID_BOTH**: both bytes of the 16 bit Data ID are used in the CRC calculation: first the low byte and then the high byte.
2. **E2E_P11_DATAID_NIBBLE**:
   - the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,
   - the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.
   - the low byte is not transmitted, but it is included in the CRC computation as start value.

[SWS_E2E_00507] In the E2E profiles 1 and 11, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data).

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

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

### 7.9.4 Length

In Profile 11 there is no explicit transmission of the length.

### 7.9.5 CRC

E2E Profile 11 uses an 8-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[SWS_E2E_00508] E2E Profile 11 shall use the `Crc_CalculateCRC8()` function of the SWS CRC Library for calculating the CRC (CRC-8-SAE J1850).

(SRS_E2E_08539, SRS_E2E_08531)

[SWS_E2E_00505] In the E2E Profile 11 with DataIDMode set to E2E_P11_DATAID_BOTH, the Data ID shall be implicitly transmitted, by adding first the Data ID low byte, then the Data ID high byte before the user data in the CRC calculation.

(SRS_E2E_08539)

[SWS_E2E_00506] In E2E Profile 11 with DataIDMode set to E2E_P01_DATAID_NIBBLE, the lower nibble of the high byte of the DataID shall be placed in the transmitted data at bit position `DataIDNibbleOffset`, and the CRC calculation shall be done by first calculating over the low byte of the Data ID, then a 0-byte, and then the user data.

(SRS_E2E_08539)

Note: the byte containing the CRC is always omitted from the CRC calculation.
7.9.6 Timeout detection

The previously mentioned mechanisms (for Profile 11: 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 UC_E2E_00089.

The attribute State->Status = E2E_P11STATUS_NONEWDATA means that the transmission medium (e.g RTE) reported that no new data element is available at the transmission medium. The attribute State->Status = E2E_P11STATUS_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 an unavailability of valid data that was updated since the previous cycle.

7.9.7 E2E_P11Protect

The function E2E_P11Protect() performs the steps as specified by the following diagrams in this section.

[SWS_E2E_00509] The function E2E_P11Protect() shall have the following overall behavior:
(SRS_E2E_08539)

[SWS_E2E_00510] The step “Verify inputs of the protect function” in E2E_P11Protect() shall have the following behavior:

- (Config != NULL) && (State != NULL) && (Data != NULL)
- Length == Config->DataLength/8

(SRS_E2E_08539)
The step “Write DataIDNibble” in E2E_P11Protect() shall have the following behavior:

Copy lower 4 bits of second byte of State->DataID to data array at position Config->DataIDNibbleOffset, using little-endian byte-order.

Byte position in data array can be obtained by:
Data[Config->DataIDNibbleOffset >> 3]

Nibble position within byte can be obtained by:
((Config->DataID & 0x0F00) >> 8) << (Config->CounterOffset & 0x7)

\[ \text{Config->DataIDMode == E2E_P11_DATAID_NIBBLE} \]

\[ \text{[FALSE]} \]

\[ \text{[TRUE]} \]

The step “Write Counter” in E2E_P11Protect() shall have the following behavior:

Copy lower 4 bits of State->Counter to data array at position Config->CounterOffset, using little-endian byte-order.

Byte position in data array can be obtained by:
Data[Config->CounterOffset >> 3]

Nibble position within byte can be obtained by:
(State->Counter & 0xF) << (Config->CounterOffset & 0x7)

\[ \text{[SRS_E2E_08539]} \]

The step “Compute CRC” in E2E_P11Protect() shall have the following behavior:
The step “Write CRC” in E2E_P11Protect() shall have the following behavior:

- Copy 1-byte local variable ComputedCRC on bytes Data[Config->CRCOffset/8]
The type is uint8, but only 15 values are used. After 0xE, the next value is 0.

7.9.8 E2E_P11Check

The function E2E_P11Check performs the actions as specified by the following six diagrams in this section.

[SWS_E2E_00516] The function E2E_P11Check() shall have the following overall behavior:
E2E_P11Check(Config, State, Data, Length)

Verify inputs of the check function

NewDataAvailable == TRUE

Read DataIDNibble

Read Counter

Read CRC

Compute CRC

Do checks

return E2E_E_INPUTERR_NULL

return E2E_E_OK

return E2E_E_INPUTERR_WRONG

(SRS_E2E_08539)

[SWS_E2E_00517] The step “Verify inputs of the check function” in E2E_P11Check() shall have the following behavior:
The step “Read Counter” in E2E_P11Check() shall have the following behavior:
The step "Read Counter" in E2E_P11Check() shall have the following behavior:

Copy nibble from bit-position Config->CounterOffset in Data in Little Endian order to uint8 local variable ReceivedCounter

Byte position in data array can be obtained by:
Byte = Data[Config->CounterOffset >> 3]
Nibble within byte can be obtained by:
Counter = (Byte << (Config->CounterOffset & 0x7)) & 0xF

The step "Read CRC" in E2E_P11Check() shall have the following behavior:

Copy byte Data[Config->CRCOffset/8] to local variable ReceivedCRC

The step "Do Checks" in E2E_P11Check() shall have the following behavior:
With the following correction:
The third decision should also evaluate to TRUE in case Config->DataIDMode == E2E_P11_DATAID_BOTH:
(DataIDMode == NIBBLE && receivedNibble == DataID>>8) || (DataIDMode != NIBBLE)
] (SRS_E2E_08539)
7.10 Specification of E2E Profile 22

Profile 22 is bus-compatible to profile 2, but provides a way of state handling similar to profiles 4 to 7 on receiver side. There is no difference in using E2E protection measures on sender side compared to profile 2.

[SWS_E2E_00522] Profile 22 shall provide the following control fields, transmitted at runtime together with the protected data:

<table>
<thead>
<tr>
<th>Control field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>4 bits. (explicitly sent)</td>
</tr>
<tr>
<td>CRC</td>
<td>8 bits, polynomial in normal form 0x2F (Autosar notation), provided by CRC library. (explicitly sent)</td>
</tr>
<tr>
<td>Data ID List</td>
<td>16 8-bit values, linked to Counter value. Effectively 16 different values, one for each counter value. The Data ID List must be unique system-wide.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>E2E Mechanism</th>
<th>Detected communication faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>Repetition, Loss, insertion, incorrect sequence, blocking</td>
</tr>
<tr>
<td>Transmission on a regular bases and timeout monitoring using E2E-Library(^1)</td>
<td>Loss, delay, blocking</td>
</tr>
<tr>
<td>Data ID + CRC</td>
<td>Masquerade and incorrect addressing, insertion</td>
</tr>
<tr>
<td>CRC</td>
<td>Corruption, Asymmetric information(^2)</td>
</tr>
</tbody>
</table>

\(^1\) Implementation by sender and receiver for a set of data protected by same CRC
\(^2\) For a set of data protected by same CRC

Table 7-8: Detectable communication faults using Profile 22

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

7.10.1 Data Layout

7.10.1.1 User data layout

In the E2E Profile 22, the user data layout (of the data to be protected) is not constrained by E2E Profile 22. The total length of transmitted data must be a multiple of 8 bit (full bytes). Also, as the header only used 12 bit, there are 4 bit unused and available for user data in the byte where the 4 bit of the counter are placed.

7.10.1.2 Header layout

Profile 22 is backward compatible to the bus-layout of profile 2. In addition, the configuration field offset can be used to offset the header fields, then breaking with backward-compatibility to profile 2 bus-layout.
7.10.2 Counter

In E2E Profile 22, the counter is initialized, incremented, reset and checked by E2E profile check and protect functions. The counter is not manipulated or used by the caller of the E2E library.

[SWS_E2E_00523] In E2E Profile 22, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0x0F), then it shall restart with 0 for the next send request.

(SRS_E2E_08539)

Note that the counter value 0x0F is not reserved as a special invalid value.

In E2E Profile 22, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:
   a. no new data has arrived since last invocation of E2E library check function,
   b. the data is repeated

2. OK:
   a. counter is incremented by one (i.e. no data lost),
   b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Error:
   a. counter is incremented more than allowed (i.e. too many data lost).
Case 1 corresponds to a failed alive counter check, and case 3 corresponds to a failed sequence counter check.

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

**7.10.3 Data ID List**

The unique Data ID List is used to verify the identity of each transmitted safety-related data element.

[SWS_E2E_00524] In the E2E profile 22, the Data IDs shall be implicitly transmitted, by taking the Data ID corresponding to the Counter value from the Data ID List and adding it after the user data in the CRC calculation.](SRS_E2E_08539)

[SWS_E2E_00525] In the E2E profiles 2 and 22, the Data ID Lists shall be globally unique within the network of communicating system (made of several ECUs each sending different data).](SRS_E2E_08539)

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

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

**7.10.4 Length**

In Profile 22 there is no explicit transmission of the data length.

**7.10.5 CRC**

E2E Profile 22 uses an 8-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance. The CRC polynomial is the same as used in profile 2.

[SWS_E2E_00526] E2E Profile 22 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Library for calculating the CRC (Polynomial 0x2F, see also SWS_E2E_00117).](SRS_E2E_08539, SRS_E2E_08531)

[SWS_E2E_00527] In E2E Profile 22, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended at the end with the corresponding Data ID from the Data ID List.](SRS_E2E_08539, SRS_E2E_08536)
7.10.6 Timeout detection

The previously mentioned mechanisms (for Profile 22: 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 UC_E2E_00089.

The attribute State->Status = E2E_P22STATUS_NONEWDATA means that the transmission medium (e.g RTE) reported that no new data element is available at the transmission medium. The attribute State->Status = E2E_P22STATUS_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 an unavailability of valid data that was updated since the previous cycle.

7.10.7 E2E_P22Protect

The function E2E_P22Protect() performs the steps as specified by the following diagrams in this section.

[SWS_E2E_00528] The function E2E_P22Protect() shall have the following overall behavior:
The step “Verify inputs of the protect function” in E2E_P22Protect() shall have the following behavior:

(SRS_E2E_08539)

[SWS_E2E_00529] The step “Write Counter” in E2E_P22Protect() shall have the following behavior:
The step “Compute CRC” in E2E_P22Protect() shall have the following behavior:

\[
\text{uint8 ComputedCRC} = \text{Crc\_CalculateCRC8H2F(Crc\_DataPtr: \&Data[1], Crc\_Length: \( \text{Length-1} \), Crc\_StartValue8: 0xFF, Crc\_IsFirstCall: TRUE)}
\]

The step “Write CRC” in E2E_P22Protect() shall have the following behavior:
The step “Increment Counter” in E2E_P22Protect() shall have the following behavior:

The type is uint8, but only 16 values are used. After 0xF, the next value is 0.

---

7.10.8 E2E_P22Check

The function E2E_P11Check performs the actions as specified by the following six diagrams in this section.

The function E2E_P22Check() shall have the following overall behavior:
The step “Verify inputs of the check function” in E2E_P22Check() shall have the following behavior:
The step “Read Counter” in E2E_P22Check() shall have the following behavior:

\[ SRS_{E2E\_08539} \]
\[ SWS_{E2E\_00536} \]
Copy 4 bit counter value from bit-position Config->Offset+8 in Data in Little Endian order to uint8 local variable Counter

Counter in data array can be obtained by:
Counter = Data[(Config->Offset >> 3)+1] & 0x0F

The step “Read CRC” in E2E_P22Check() shall have the following behavior:

Copy byte Data[Config->Offset/8] to local variable ReceivedCRC

The step “Do Checks” in E2E_P22Check() shall have the following behavior:
7.11 Specification of E2E state machine

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

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

7.11.1 Overview of the state machine

The diagram below summarizes the state machine.

![State Machine Diagram]

7.11.2 State machine specification

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

[SWS_E2E_00345]: The E2E State machine shall have the following behavior with
respect to the function E2E_SMCheck():
This shall be understood as follows:

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

(SRS_E2E_08539)

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

State->ProfileStatusWindow[State->WindowTopIndex] = ProfileStatus

State->OKCount = number of elements in State->ProfileStatusWindow[] with values E2E_P_OK

State->ErrorCount = number of elements in State->ProfileStatusWindow[] with values E2E_P_ERROR

If (State->WindowTopIndex == Config->WindowSize -1) then State->WindowTopIndex=0, else State->WindowTopIndex++

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

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

} (SRS_E2E_08539)
The step E2E_SMClearStatus(ProfileStatus, State) in E2E_SMCheck() shall have the following behavior:

[SWS_E2E_00467]
### 7.12 Version Check

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

For included header files:
- E2E_AR_RELEASE_MAJOR_VERSION
- E2E_AR_RELEASE_MINOR_VERSION

shall be identical. For the module internal c and h files:
- E2E_SW_MAJOR_VERSION
- E2E_SW_MINOR_VERSION
- E2E_AR_RELEASE_MAJOR_VERSION
- E2E_AR_RELEASE_MINOR_VERSION
- E2E_AR_RELEASE_REVISION_VERSION

```c
E2E_SMClearStatus(ProfileStatus, State)
State->ProfileStatusWindow[] = E2E_P_NOTAVAILABLE
State->OKCount = 0
State->ErrorCount = 0
State->WindowTopIndex = 0
```

Clear each element of the ProfileStatusWindow[] array.
shall be identical (see also [SWS_E2E_00038] for published information).
(SRS_BSW_00003)
8 API specification

This chapter specifies the API of E2E Library.

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

8.1 Imported types

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Imported Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std_Types</td>
<td>Std_ReturnType</td>
</tr>
<tr>
<td></td>
<td>Std_VersionInfoType</td>
</tr>
</tbody>
</table>

(SRS_E2E_08528)

8.2 Type definitions

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

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

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

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

Example 1 - Counter with bit offset = 8 on MSB microcontroller:
### 8.2.1 E2E Profile 1 types

Note: Since AUTOSAR 4.1.1, type names were renamed. If an existing application using E2E Library requires compatibility of interfaces to previous release versions, then the header file E2E_P01.h shall contain following type definitions:

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

#### Figure 8-1: E2E Profile 1 configuration

### 8.2.1.1 E2E_P01ConfigType

**[SWS_E2E_00018]**

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P01ConfigType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Structure</td>
</tr>
<tr>
<td>Element:</td>
<td></td>
</tr>
<tr>
<td>uint16</td>
<td>CounterOffset</td>
</tr>
<tr>
<td>uint16</td>
<td>CRCOffset</td>
</tr>
<tr>
<td>uint16</td>
<td>DataID</td>
</tr>
</tbody>
</table>
|     uint16      | DataIDNibbleOffset      | Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P01_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than E2E_P01_DATAID_NIBBLE, DataIDNibbleOffset shall be initialized to 0 (even if it is
### Specification of SW-C End-to-End Communication Protection Library

**AUTOSAR CP Release 4.3.1**

#### 8.2.1.2 E2E_P01DataIDMode

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P01DataIDMode</td>
<td>Enumeration</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P01_DATAID_BOTH</td>
<td>Two bytes are included in the CRC (double ID configuration). This is used in E2E variant 1A.</td>
</tr>
<tr>
<td>E2E_P01_DATAID_ALT</td>
<td>One of the two bytes is included, alternating high and low byte, depending on parity of the counter (alternating ID configuration). For an even counter, the low byte is included. For an odd counter, the high byte is included. This is used in E2E variant 1B.</td>
</tr>
<tr>
<td>E2E_P01_DATAID_LOW</td>
<td>Only the low byte is included, the high byte is never used. This is applicable if the IDs in a particular system Ignored by E2E Library).</td>
</tr>
</tbody>
</table>

**Description:** Configuration of transmitted Data (Data Element or IPDU), for E2E Profile 1. For each transmitted Data, there is an instance of this typedef.
8.2.1.3 E2E_P01ProtectStateType

**Description:**
The Data ID is two bytes long in E2E Profile 1. There are four inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.

<table>
<thead>
<tr>
<th>Name: E2E_P01ProtectStateType</th>
<th>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.</th>
</tr>
</thead>
</table>

8.2.1.4 E2E_P01CheckStateType

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

![UML Diagram](image)

**Description:**
State of the sender for a Data protected with E2E Profile 1.

Figure 8-2: E2E Profile 1 check state type
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>MaxDeltaCounter</td>
<td>MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.</td>
</tr>
<tr>
<td>boolean</td>
<td>WaitForFirstData</td>
<td>If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.</td>
</tr>
<tr>
<td>boolean</td>
<td>NewDataAvailable</td>
<td>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.</td>
</tr>
<tr>
<td>uint8</td>
<td>LostData</td>
<td>Number of data (messages) lost since reception of last valid one. This attribute is set only if Status equals E2E_P01STATUS_OK or E2E_P01STATUS_OKSOMELOST. For other values of Status, the value of LostData is undefined. E2E_P01CheckStatusType Status Result of the verification of the Data, determined by the Check function.</td>
</tr>
<tr>
<td>E2E_P01CheckStatusType</td>
<td>Status</td>
<td>Result of the verification of the Data, determined by the Check function.</td>
</tr>
<tr>
<td>uint8</td>
<td>SyncCounter</td>
<td>Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.</td>
</tr>
<tr>
<td>uint8</td>
<td>NoNewOrRepeatedDataCounter</td>
<td>Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.</td>
</tr>
</tbody>
</table>

**Description:** State of the receiver for a Data protected with E2E Profile 1. 

(SRS_E2E_08528, SRS_E2E_08534)

### 8.2.1.5 E2E_P01CheckStatusType

<table>
<thead>
<tr>
<th>Name: E2E_P01CheckStatusType</th>
<th>Type: Enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P01STATUS_OK</td>
<td>0x00 OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.</td>
</tr>
<tr>
<td>E2E_P01STATUS_NONEWDATA</td>
<td>0x01 Error: the Check function has been called without new data.</td>
</tr>
</tbody>
</table>
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\_P01\_STATUS\_WRONGCRC**

0x02 Error: The data has been received according to communication medium, but
1. the CRC is incorrect (applicable for all E2E Profile 1 configurations) or
2. the low nibble of the high byte of Data ID is incorrect (applicable only for E2E Profile 1 with E2E\_P01\_DataIDMode = E2E\_P01\_DATAID\_NIBBLE).

The two above errors can be a result of corruption, incorrect addressing or masquerade.

**E2E\_P01\_STATUS\_SYNC**

0x03 NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.

**E2E\_P01\_STATUS\_INITIAL**

0x04 Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.

**E2E\_P01\_STATUS\_REPEATED**

0x08 Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.

**E2E\_P01\_STATUS\_OKSOMELOST**

0x20 OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.

**E2E\_P01\_STATUS\_WRONGSEQUENCE**

0x40 Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
### Description:
Result of the verification of the Data in E2E Profile 1, determined by the Check function.

[SRS_E2E_08534]

## 8.2.2 E2E Profile 2 types

Since AUTOSAR 4.1.1, type names were renamed. If an existing application using E2E Library requires compatibility of interfaces to previous release versions, then the header file E2E_P02.h shall contain following type definitions:

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

### 8.2.2.1 E2E_P02ConfigType

[SWS_E2E_00152]

<table>
<thead>
<tr>
<th>Name</th>
<th>E2E_P02ConfigType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Structure</td>
</tr>
<tr>
<td>Element</td>
<td></td>
</tr>
<tr>
<td><code>uint16</code></td>
<td>DataLength</td>
</tr>
<tr>
<td>Description</td>
<td>Length of Data, in bits. The value shall be a multiple of 8.</td>
</tr>
</tbody>
</table>

| `uint8[16]`   | DataIDList                                 |
| Description   | An array of appropriately chosen Data IDs for protection against masquerading. |

| `uint8`       | MaxDeltaCounterInit                       |
| Description   | Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1. |

| `uint8`       | MaxNoNewOrRepeatedData                    |
| Description   | The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions. |

| `uint8`       | SyncCounterInit                           |
| Description   | Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter. |

| `uint16`      | Offset                                    |
| Description   | Offset of the E2E header in the Data[] array in bits. It shall be: 0 ≤ Offset ≤ DataLength-(2*8). |

| Description   | Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 2. |
The position of the counter and CRC is not configurable in profile 2.

8.2.2.2 E2E_P02ProtectStateType

<table>
<thead>
<tr>
<th>Name: E2E_P02ProtectStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Structure</td>
</tr>
<tr>
<td>Element: uint8 Counter</td>
</tr>
<tr>
<td>Description: State of the sender for a Data protected with E2E Profile 2.</td>
</tr>
</tbody>
</table>

8.2.2.3 E2E_P02CheckStateType

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

Figure 8-3: E2E Profile 2 check state

<table>
<thead>
<tr>
<th>Name: E2E_P02CheckStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Structure</td>
</tr>
<tr>
<td>Element: uint8 LastValidCounter</td>
</tr>
<tr>
<td>Description: Counter of last valid received message.</td>
</tr>
</tbody>
</table>

| Element: uint8 MaxDeltaCounter |
| Description: MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages. |

<p>| Element: boolean WaitForFirstData |
| Description: If true means that no correct data (with correct Data ID and |</p>
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>NewDataAvailable</td>
<td>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.</td>
</tr>
<tr>
<td>uint8</td>
<td>LostData</td>
<td>Number of data (messages) lost since reception of last valid one.</td>
</tr>
<tr>
<td>E2E_P02CheckStatusType</td>
<td>Status</td>
<td>Result of the verification of the Data, determined by the Check function.</td>
</tr>
<tr>
<td>uint8</td>
<td>SyncCounter</td>
<td>Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.</td>
</tr>
<tr>
<td>uint8</td>
<td>NoNewOrRepeatedDataCounter</td>
<td>Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.</td>
</tr>
</tbody>
</table>

**Description:** State of the sender for a Data protected with E2E Profile 2.

(SRS_E2E_08528, SRS_E2E_08534)

### 8.2.2.4 E2E_P02CheckStatusType

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

[SWS_E2E_00214]|
<p>| Name: | E2E_P02CheckStatusType |</p>
<table>
<thead>
<tr>
<th>Type:</th>
<th>Enumeration</th>
<th></th>
<th>Range:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E2E_P02STATUS_OK</td>
<td>0x00</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_NONEWDATA</td>
<td>0x01</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_WRONGCRC</td>
<td>0x02</td>
<td>Error: The data has been received according to communication medium, but the CRC is incorrect.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_SYNC</td>
<td>0x03</td>
<td>NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_INITIAL</td>
<td>0x04</td>
<td>Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_REPEATED</td>
<td>0x08</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_OKSOMELOST</td>
<td>0x20</td>
<td>OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 &lt; 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E2E_P02STATUS_WRONGSEQUENCE</td>
<td>0x40</td>
<td>Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter &gt; 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</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Specification of SW-C End-to-End
Communication Protection Library
AUTOSAR CP Release 4.3.1

<table>
<thead>
<tr>
<th>Description:</th>
<th>Result of the verification of the Data in E2E Profile 2, determined by the Check function.</th>
</tr>
</thead>
</table>

[SRS_E2E_08534]

8.2.3 E2E Profile 4 types

Figure 8-4: E2E Profile 4 configuration

8.2.3.1 E2E_P04ConfigType

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P04ConfigType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>DataID</td>
</tr>
<tr>
<td></td>
<td>A system-unique identifier of the Data.</td>
</tr>
<tr>
<td>uint16</td>
<td>Offset</td>
</tr>
<tr>
<td></td>
<td>Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ MaxDataLength - (12·8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.</td>
</tr>
<tr>
<td>uint16</td>
<td>MinDataLength</td>
</tr>
<tr>
<td></td>
<td>Minimal length of Data, in bits. E2E checks that Length is ≥ MinDataLength. The value shall be = 4096·8 (4kB) and shall be ≥ 12·8</td>
</tr>
<tr>
<td>uint16</td>
<td>MaxDataLength</td>
</tr>
<tr>
<td></td>
<td>Maximal length of Data, in bits. E2E checks that DataLength is ≤ MinDataLength. The value shall be = 4096·8 (4kB) and it shall be ≥ MinDataLength</td>
</tr>
<tr>
<td>uint16</td>
<td>MaxDeltaCounter</td>
</tr>
<tr>
<td></td>
<td>Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept the last correct/initial reception.</td>
</tr>
</tbody>
</table>

[SWS_E2E_00334]
### 8.2.3.2 E2E_P04ProtectStateType

**[SWS_E2E_00335]**

<table>
<thead>
<tr>
<th>Name: E2E_P04ProtectStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Structure</td>
</tr>
<tr>
<td>Element: uint16 Counter</td>
</tr>
</tbody>
</table>

Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P04Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.

**Description:** State of the sender for a Data protected with E2E Profile 4.

### 8.2.3.3 E2E_P04CheckStateType

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

![Figure 8-5: E2E Profile 4 check state](image)

**[SWS_E2E_00336]**

<table>
<thead>
<tr>
<th>Name: E2E_P04CheckStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Structure</td>
</tr>
<tr>
<td>Element: E2E_P04CheckStatusType Status</td>
</tr>
</tbody>
</table>

Result of the verification of the Data in this cycle, determined by the Check function.

**Description:** State of the reception on one single Data protected with E2E Profile 4.

### 8.2.3.4 E2E_P04CheckStatusType

**[SWS_E2E_00337]**

<table>
<thead>
<tr>
<th>Name: E2E_P04CheckStatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Structure</td>
</tr>
<tr>
<td>Element: E2E_P04CheckStatusType Status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counter :uint16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status :E2E_P04CheckStatusType</td>
</tr>
</tbody>
</table>

Figure 8-5: E2E Profile 4 check state.
Specification of SW-C End-to-End Communication Protection Library
AUTOSAR CP Release 4.3.1

Name: E2E_P04CheckStatusType
Type: --

<table>
<thead>
<tr>
<th>Range:</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P04STATUS_OK</td>
<td>0x00</td>
<td>OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).</td>
</tr>
<tr>
<td>E2E_P04STATUS_NONEWDATA</td>
<td>0x03</td>
<td>Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P04STATUS_REPEATED.</td>
</tr>
<tr>
<td>E2E_P04STATUS_ERROR</td>
<td>0x02</td>
<td>Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).</td>
</tr>
<tr>
<td>E2E_P04STATUS_REPEATED</td>
<td>0x06</td>
<td>Error: the checks of the Data in this cycle were successful, with the exception of the repetition.</td>
</tr>
<tr>
<td>E2E_P04STATUS_OKSOMELOST</td>
<td>0x20</td>
<td>OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).</td>
</tr>
<tr>
<td>E2E_P04STATUS_WRONGSEQUENCE</td>
<td>0x40</td>
<td>Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.</td>
</tr>
</tbody>
</table>

Description: Status of the reception on one single Data in one cycle, protected with E2E Profile 4.

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

8.2.4 E2E Profile 5 types

8.2.4.1 E2E_P05ConfigType

```cpp
«structure»
E2E::E2E_P05Types::
E2E_P05ConfigType
+ DataID :uint16
+ DataLength :uint16
+ MaxDeltaCounter :uint8
+ Offset :uint16
```

Figure 8-6: E2E Profile 5 configuration
### Type: Structure

### Element:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>Offset</td>
<td>Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ DataLength-(3*8). Example: If Offset equals 8, then the low byte of the E2E Crc (16 bit) is written to Byte 1, the high Byte is written to Byte 2.</td>
</tr>
<tr>
<td>uint16</td>
<td>DataLength</td>
<td>Length of Data, in bits. The value shall be = 4096<em>8 (4kB) and shall be ≥ 3</em>8</td>
</tr>
<tr>
<td>uint16</td>
<td>DataID</td>
<td>A system-unique identifier of the Data</td>
</tr>
<tr>
<td>uint8</td>
<td>MaxDeltaCounter</td>
<td>Maximum allowed gap between two consecutive Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.</td>
</tr>
</tbody>
</table>

### Description:

Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 5. For each transmitted Data, there is an instance of this typedef.

---

### 8.2.4.2 E2E_P05ProtectStateType

**Figure 8-7: E2E Profile 5 Protect state type**

[SWS_E2E_00438] |

- **Name:** E2E_P05ProtectStateType
- **Type:** Structure
- **Element:**
  - uint8 Counter
  - Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P05Protect() is called, it increments the counter up to 0xFF.

**Description:** State of the sender for a Data protected with E2E Profile 5.

[SRS_E2E_08539]

---

### 8.2.4.3 E2E_P05CheckStateType

Note: The values for the enumeration constants are specified only on the associated UML diagram (not in the table).
Figure 8-8: E2E Profile 5 Check state type

[SWS_E2E_00439]

| Name: | E2E_P05CheckStateType |
| Type: | Structure |
| Element: | E2E_P05CheckStatusType Status |
| Description: | Result of the verification of the Data in this cycle, determined by the Check function. |
| Description: | State of the reception on one single Data protected with E2E Profile 5. |

(SRS_E2E_08539, SRS_E2E_08534)
8.2.4.4 E2E_P05CheckStatusType

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P05CheckStatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P05STATUS_OK</td>
<td>0x00</td>
</tr>
<tr>
<td>E2E_P05STATUS_NONEWDATA</td>
<td>0x01</td>
</tr>
<tr>
<td>E2E_P05STATUS_ERROR</td>
<td>0x07</td>
</tr>
<tr>
<td>E2E_P05STATUS_REPEATED</td>
<td>0x08</td>
</tr>
<tr>
<td>E2E_P05STATUS_OKSOMELOST</td>
<td>0x20</td>
</tr>
<tr>
<td>E2E_P05STATUS_WRONGSEQUENCE</td>
<td>0x40</td>
</tr>
</tbody>
</table>

Description: Status of the reception on one single Data in one cycle, protected with E2E Profile 5.
8.2.5  E2E Profile 6 types

8.2.5.1  E2E_P06ConfigType

```plaintext
«structure»
E2E::E2E_P06Types::
E2E_P06ConfigType
+ DataID :uint16
+ MaxDataLength :uint16
+ MaxDeltaCounter :uint8
+ MinDataLength :uint16
+ Offset :uint16
```

Figure 8-9: E2E Profile 6 configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>E2E_P06ConfigType</th>
<th>Type</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| uint16             | Offset                       | Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ MaxDataLength−(5*8).
|                    | Example: If Offset equals 8, then the high byte of the E2E Crc (16 bit) is written to Byte 1, the low Byte is written to Byte 2. |
| uint16             | MinDataLength                | Minimal length of Data, in bits (i.e. the length of the E2E-protected message, without counting the field Length itself). E2E checks that Length is ≥ MinDataLength. The value shall be = 4096*8 (4kB) and shall be ≥ 5*8 |
| uint16             | MaxDataLength                | Maximal length of Data, in bits. E2E checks that DataLength is ≤ MaxDataLength. The value shall be = 4096*8 (4kB). |
| uint16             | DataID                       | A system-unique identifier of the Data |
| uint8              | MaxDeltaCounter              | Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4. |

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

(SRS_E2E_08539, SRS_E2E_08534)

159 of 315

- AUTOSAR confidential -
8.2.5.2 E2E_P06ProtectStateType

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P06ProtectStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Structure</td>
</tr>
<tr>
<td>Element:</td>
<td>uint8  Counter</td>
</tr>
<tr>
<td>Description:</td>
<td>State of the sender for a Data protected with E2E Profile 6.</td>
</tr>
</tbody>
</table>

Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P06Protect() is called, it increments the counter up to 0xFF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.

(SRS_E2E_08539)
8.2.5.3 E2E_P06CheckStateType

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

![Diagram of E2E_P06CheckStateType structure]

Figure 8-11: E2E Profile 6 Check state type

[SW_S2E_00444] | 
--- | --- | 
**Name:** E2E_P06CheckStateType | 
**Type:** Structure | 
**Element:** E2E_P06CheckStatusType::Status | 
**Description:** State of the reception on one single Data protected with E2E Profile 6.

| (SRS_E2E_08539, SRS_E2E_08534) | 
--- | --- | 
+ Counter : uint8 | 
+ Status : E2E_P06CheckStatusType | 
+ E2E_P06STATUS_ERROR = 0x07 | 
+ E2E_P06STATUS_NONEWDATA = 0x01 | 
+ E2E_P06STATUS_OK = 0x00 | 
+ E2E_P06STATUS_OKSOMELOST = 0x20 | 
+ E2E_P06STATUS_REPEATED = 0x08 | 
+ E2E_P06STATUS_WRONGSEQUENCE = 0x40 |
### 8.2.5.4 E2E_P06CheckStatusType

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P06CheckStatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
<td></td>
</tr>
<tr>
<td>E2E_P06STATUS_OK</td>
<td>OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).</td>
</tr>
<tr>
<td>E2E_P06STATUS_NONEWDATA</td>
<td>Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P06STATUS_REPEATED.</td>
</tr>
<tr>
<td>E2E_P06STATUS_ERROR</td>
<td>Error: error not related to counters occurred (e.g. wrong crc, wrong length).</td>
</tr>
<tr>
<td>E2E_P06STATUS_REPEATED</td>
<td>Error: the checks of the Data in this cycle were successful, with the exception of the repetition.</td>
</tr>
<tr>
<td>E2E_P06STATUS_OKSOMELOST</td>
<td>OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).</td>
</tr>
<tr>
<td>E2E_P06STATUS_WRONGSEQUENCE</td>
<td>Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.</td>
</tr>
</tbody>
</table>

**Description:** Status of the reception on one single Data in one cycle, protected with E2E Profile 6.
8.2.6 E2E Profile 7 types

8.2.6.1 E2E_P07ConfigType

```
«structure»
E2E::E2E_P07Types::
E2E_P07ConfigType
+ DataID ::uint32
+ MaxDataLength ::uint32
+ MaxDeltaCounter ::uint32
+ MinDataLength ::uint32
+ Offset ::uint32
```

Figure 8-12: E2E Profile 7 configuration

[SWS_E2E_00544] [SRS_E2E_08539]

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P07ConfigType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>DataID</td>
<td>A system-unique identifier of the Data.</td>
</tr>
<tr>
<td>uint32</td>
<td>Offset</td>
<td>Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ MaxDataLength- (20*8). Example: If Offset equals 8, then the first byte of the E2E Length (32 bit) is written to byte 1, the next byte is written to byte 2 and so on.</td>
</tr>
<tr>
<td>uint32</td>
<td>MinDataLength</td>
<td>Minimal length of Data, in bits. E2E checks that Length is ≥ MinDataLength. The value shall be ≥ 20*8 and ≤ MaxDataLength..</td>
</tr>
<tr>
<td>uint32</td>
<td>MaxDataLength</td>
<td>Maximal length of Data, in bits. E2E checks that DataLength is ≤ MinDataLength. The value shall be ≥ MinDataLength</td>
</tr>
<tr>
<td>uint32</td>
<td>MaxDeltaCounter</td>
<td>Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.</td>
</tr>
</tbody>
</table>

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

[SRS_E2E_08539]
8.2.6.2 E2E_P07ProtectStateType

```plaintext
«structure»
E2E::E2E_P07Types::
E2E_P07ProtectStateType
+ Counter : uint32
```

Figure 8-13: E2E Profile 7 Protect state type

[SWS_E2E_00545]

<table>
<thead>
<tr>
<th>Name</th>
<th>E2E_P07ProtectStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Structure</td>
</tr>
<tr>
<td>Element: Counter</td>
<td>uint32</td>
</tr>
<tr>
<td>Description:</td>
<td>State of the sender for a Data protected with E2E Profile 7.</td>
</tr>
</tbody>
</table>

(SRS_E2E_08539)

8.2.6.3 E2E_P07CheckStateType

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

```plaintext
«structure»
E2E::E2E_P07Types::
E2E_P07CheckStateType
+ Counter : uint32
+ Status : E2E_P07CheckStatusType
```

```plaintext
«type»
E2E::E2E_P07Types::E2E_P07CheckStatusType
+ E2E_P07STATUS_ERROR = 0x02
+ E2E_P07STATUS_NENWDATA = 0x01
+ E2E_P07STATUS_OK = 0x00
+ E2E_P07STATUS_OKSOMELOST = 0x20
+ E2E_P07STATUS_REPEATED = 0x08
+ E2E_P07STATUS_WRONGSEQUENCE = 0x40
```

Figure 8-14: E2E Profile 7 Check state type

[SWS_E2E_00542]

<table>
<thead>
<tr>
<th>Name</th>
<th>E2E_P07CheckStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Structure</td>
</tr>
<tr>
<td>Element: Status</td>
<td>E2E_P07CheckStatusType</td>
</tr>
<tr>
<td>Description:</td>
<td>State of the reception on one single Data protected with E2E Profile 7.</td>
</tr>
</tbody>
</table>

(SRS_E2E_08539, SRS_E2E_08534)
### 8.2.6.4 E2E_P07CheckStatusType

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P07CheckStatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
<td></td>
</tr>
<tr>
<td>E2E_P07STATUS_OK</td>
<td>0x00</td>
</tr>
<tr>
<td><strong>OK:</strong> the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).</td>
<td></td>
</tr>
<tr>
<td>E2E_P07STATUS_NONEWDATA</td>
<td>0x01</td>
</tr>
<tr>
<td><strong>Error:</strong> the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P07STATUS_REPEATED.</td>
<td></td>
</tr>
<tr>
<td>E2E_P07STATUS_ERROR</td>
<td>0x02</td>
</tr>
<tr>
<td><strong>Error:</strong> error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).</td>
<td></td>
</tr>
<tr>
<td>E2E_P07STATUS_REPEATED</td>
<td>0x08</td>
</tr>
<tr>
<td><strong>Error:</strong> the checks of the Data in this cycle were successful, with the exception of the repetition.</td>
<td></td>
</tr>
<tr>
<td>E2E_P07STATUS_OKSOMELOST</td>
<td>0x20</td>
</tr>
<tr>
<td><strong>OK:</strong> the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).</td>
<td></td>
</tr>
<tr>
<td>E2E_P07STATUS_WRONGSEQUENCE</td>
<td>0x40</td>
</tr>
<tr>
<td><strong>Error:</strong> the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.</td>
<td></td>
</tr>
</tbody>
</table>

**Description:** Status of the reception on one single Data in one cycle, protected with E2E Profile 7.

(SRS_E2E_08534)
### 8.2.7 E2E Profile 11 types

#### 8.2.7.1 E2E_P11ConfigType

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P11ConfigType</td>
<td>Structure</td>
<td>Length of data, in bits. The value shall be a multiple of 8 and shall be ≤ 240.</td>
</tr>
<tr>
<td>E2E_P11ConfigType</td>
<td>Structure</td>
<td>A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section &quot;Configuration constraints on Data IDs&quot;.</td>
</tr>
<tr>
<td>E2E_P11ConfigType</td>
<td>Structure</td>
<td>Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.</td>
</tr>
<tr>
<td>E2E_P11ConfigType</td>
<td>Structure</td>
<td>Bit offset of CRC (i.e. since &quot;Data&quot;) in MSB first order. In variants 1A and 1B, CRCOffset is 0. The offset shall be a multiple of 8.</td>
</tr>
<tr>
<td>E2E_P11ConfigType</td>
<td>Structure</td>
<td>Bit offset of Counter in MSB first order. In variants 1A and 1B, CounterOffset is 8. The offset shall be a multiple of 4.</td>
</tr>
<tr>
<td>E2E_P11ConfigType</td>
<td>Structure</td>
<td>Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P01_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than</td>
</tr>
</tbody>
</table>
8.2.7.2 E2E_P11ProtectStateType

| «structure» | E2E::E2E_P11Types::E2E_P11ProtectStateType |
| + Counter :uint8 |

Figure 8-16: E2E Profile 11 Protect state type

[SWS_E2E_00567] |

Name: E2E_P11ProtectStateType
Type: Structure
Element: uint8 Counter
Description: Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P05Protect() is called, it increments the counter up to 0xFF.

8.2.7.3 E2E_P11CheckStateType

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

| «type» | E2E::E2E_P11Types::E2E_P11CheckStatusType |
| + E2E_P11STATUS_ERROR = 0x07 |
| + E2E_P11STATUS_NOWDATA = 0x01 |
| + E2E_P11STATUS_OK = 0x00 |
| + E2E_P11STATUS_OKSOMELOST = 0x20 |
| + E2E_P11STATUS_REPEATED = 0x08 |
| + E2E_P11STATUS_WRONGSEQUENCE = 0x40 |

Figure 8-17: E2E Profile 11 Check state type

[SWS_E2E_00563] |

Name: E2E_P11CheckStateType
Type: Structure
Element: E2E_P11CheckStatusType Status
Description: Result of the verification of the Data in this cycle, determined by the
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>Counter of the data in previous cycle.</td>
</tr>
</tbody>
</table>

Description: State of the reception on one single Data protected with E2E Profile 11.

(SRS_E2E_08539, SRS_E2E_08534)
### 8.2.7.4 E2E_P11CheckStatusType

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_P11CheckStatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>--</td>
</tr>
<tr>
<td>Range:</td>
<td></td>
</tr>
<tr>
<td>E2E_P11STATUS_OK</td>
<td>0x00</td>
</tr>
<tr>
<td>E2E_P11STATUS_NONEWDATA</td>
<td>0x01</td>
</tr>
<tr>
<td>E2E_P11STATUS_ERROR</td>
<td>0x07</td>
</tr>
<tr>
<td>E2E_P11STATUS_REPEATED</td>
<td>0x08</td>
</tr>
<tr>
<td>E2E_P11STATUS_OKSOMELOST</td>
<td>0x20</td>
</tr>
<tr>
<td>E2E_P11STATUS_WRONGSEQUENCE</td>
<td>0x40</td>
</tr>
</tbody>
</table>

**Description:** Status of the reception on one single Data in one cycle, protected with E2E Profile 11.

(SRS_E2E_08539)
### 8.2.8 E2E Profile 22 types

#### 8.2.8.1 E2E_P22ConfigType

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

Figure 8-18: E2E Profile 22 configuration
8.2.8.2 E2E_P22ProtectStateType

![Diagram of E2E_P22ProtectStateType]

**Figure 8-19: E2E Profile 22 Protect state type**

[SWS_E2E_00570]

<table>
<thead>
<tr>
<th>Name</th>
<th>E2E_P22ProtectStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Structure</td>
</tr>
<tr>
<td>Element: Counter</td>
<td>uint8</td>
</tr>
<tr>
<td>Description:</td>
<td>State of the sender for a Data protected with E2E Profile 22.</td>
</tr>
</tbody>
</table>

(SRS_E2E_08528)

8.2.8.3 E2E_P22CheckStateType

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

![Diagram of E2E_P22CheckStateType]

**Figure 8-20: E2E Profile 22 Check state type**

[SWS_E2E_00568]

<table>
<thead>
<tr>
<th>Name</th>
<th>E2E_P22CheckStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Structure</td>
</tr>
<tr>
<td>Element: Counter</td>
<td>uint8</td>
</tr>
<tr>
<td>Element: Status</td>
<td>E2E_P22CheckStatusType</td>
</tr>
<tr>
<td>Description:</td>
<td>State of the sender for a Data protected with E2E Profile 22.</td>
</tr>
</tbody>
</table>

(SRS_E2E_08528, SRS_E2E_08534)
### 8.2.8.4 E2E_P22CheckStatusType

<table>
<thead>
<tr>
<th>[SWS_E2E_00569]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name:</strong></td>
<td>E2E_P22CheckStatusType</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
<td>--</td>
</tr>
<tr>
<td><strong>Range:</strong></td>
<td></td>
</tr>
<tr>
<td>E2E_P22STATUS_OK</td>
<td>0x00</td>
</tr>
<tr>
<td>E2E_P22STATUS_NONEWDATA</td>
<td>0x01</td>
</tr>
<tr>
<td>E2E_P22STATUS_ERROR</td>
<td>0x07</td>
</tr>
<tr>
<td>E2E_P22STATUS_REPEATED</td>
<td>0x08</td>
</tr>
<tr>
<td>E2E_P22STATUS_OKSOMELOST</td>
<td>0x20</td>
</tr>
<tr>
<td>E2E_P22STATUS_WRONGSEQUENCE</td>
<td>0x40</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Result of the verification of the Data in E2E Profile 22, determined by the Check function.</td>
</tr>
</tbody>
</table>

*(SRS_E2E_08534)*
8.2.9 E2E state machine types

8.2.9.1 E2E_PCheckStatusType

[SWS_E2E_00347]

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_PCheckStatusType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>--</td>
</tr>
<tr>
<td>Range:</td>
<td></td>
</tr>
<tr>
<td>E2E_P_OK</td>
<td>0x00</td>
</tr>
<tr>
<td>E2E_P_REPEATED</td>
<td>0x01</td>
</tr>
<tr>
<td>E2E_P_WRONGSEQUENCE</td>
<td>0x02</td>
</tr>
<tr>
<td>E2E_P_ERROR</td>
<td>0x03</td>
</tr>
<tr>
<td>E2E_P_NOTAVAILABLE</td>
<td>0x04</td>
</tr>
<tr>
<td>E2E_P_NONEWDATA</td>
<td>0x05</td>
</tr>
<tr>
<td>reserved</td>
<td>0x07, 0x0F</td>
</tr>
</tbody>
</table>

Description: Profile-independent status of the reception on one single Data in one cycle.

[SRS_E2E_08539]

8.2.9.2 E2E_SMConfigType

[SWS_E2E_00342]

<table>
<thead>
<tr>
<th>Name:</th>
<th>E2E_SMConfigType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Structure</td>
</tr>
<tr>
<td>Element:</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td>WindowSize</td>
</tr>
<tr>
<td>uint8</td>
<td>MinOkStateInit</td>
</tr>
<tr>
<td>uint8</td>
<td>MaxErrorStateInit</td>
</tr>
<tr>
<td>uint8</td>
<td>MinOkStateValid</td>
</tr>
<tr>
<td>uint8</td>
<td>MaxErrorStateValid</td>
</tr>
</tbody>
</table>
### 8.2.9.3 E2E_SMCheckStateType

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

- **E2E_SMCheckStateType**
  - **ErrorCount**: uint8
  - **OkCount**: uint8
  - **ProfileStatusWindow**: uint8*
  - **SMState**: E2E_SMStateType
  - **WindowTopIndex**: uint8

Figure 8-21: E2E SM check state

[SWS_E2E_00343] |
### 8.2.9.4 E2E_SMStateType

<table>
<thead>
<tr>
<th>Name</th>
<th>Type:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_SM_VALID</td>
<td>0x00</td>
<td>Communication functioning properly according to E2E, data can be used.</td>
</tr>
<tr>
<td>E2E_SM_DEINIT</td>
<td>0x01</td>
<td>State before E2E_SMCheckInit() is invoked, data cannot be used.</td>
</tr>
<tr>
<td>E2E_SM_NODATA</td>
<td>0x02</td>
<td>No data from the sender is available since the initialization, data cannot be used.</td>
</tr>
<tr>
<td>E2E_SM_INIT</td>
<td>0x03</td>
<td>There has been some data received since startup, but it is not yet possible to use it, data cannot be used.</td>
</tr>
<tr>
<td>E2E_SM_INVALID</td>
<td>0x04</td>
<td>Communication not functioning properly, data cannot be used.</td>
</tr>
<tr>
<td>reserved</td>
<td>0x07, 0x0F</td>
<td>Reserved for runtime errors (shall not be used for any state in future)</td>
</tr>
</tbody>
</table>

**Description:** Status of the communication channel exchanging the data. If the status is OK, then the data may be used.
8.3 Routine definitions

This chapter defines the routines provided by E2E Library. The provided routines can be implemented as:
1. Functions
2. Inline functions
3. Macros

8.3.1 E2E Profile 1 routines

8.3.1.1 E2E_P01Protect

[SWS_E2E_00166] |
| Service name:  | E2E_P01Protect |
| Syntax:        | Std_ReturnType E2E_P01Protect(  
|               |     const E2E_P01ConfigType* ConfigPtr,  
|               |     E2E_P01ProtectStateType* StatePtr,  
|               |     uint8* DataPtr  
|               | ) |
| Service ID[hex]: | 0x01 |
| Sync/Async:     | Synchronous |
| Reentrancy:     | Reentrant |
| Parameters (in): | ConfigPtr Pointer to static configuration. |
| Parameters (in/out): | StatePtr Pointer to port/data communication state. |
| Parameters (out): | DataPtr Pointer to Data to be transmitted. |
| Return value:   | Std_ReturnType E2E_E_INPUTERR_NULL  
|                 | E2E_E_INPUTERR_WRONG  
|                 | E2E_E_INTERR  
|                 | E2E_E_OK  
|                 | For definitions for return values, see SWS_E2E_00047. |
| Description:    | Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID. |

] (SRS_E2E_08528, SRS_E2E_08527)
8.3.1.2 E2E_P01ProtectInit

[SWS_E2E_00385] [ ]

**Service name:** E2E_P01ProtectInit

**Syntax:**

```c
Std_ReturnType E2E_P01ProtectInit(
    E2E_P01ProtectStateType* StatePtr
)
```

**Service ID[hex]:** 0x1b

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):** None

**Parameters (inout):** StatePtr  Pointer to port/data communication state.

**Return value:**

- Std_ReturnType E2E_E_INPUTERR_NULL - null pointer passed
- E2E_E_OK

**Description:**

Initializes the protection state.

[SWS_E2E_00386] [ ]

In case State is NULL, E2E_P01ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0. (SRS_E2E_08528)

8.3.1.3 E2E_P01Check

[SWS_E2E_00158] [ ]

**Service name:** E2E_P01Check

**Syntax:**

```c
Std_ReturnType E2E_P01Check(
    const E2E_P01ConfigType* Config,
    E2E_P01CheckStateType* State,
    const uint8* Data
)
```

**Service ID[hex]:** 0x02

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):**

- Config  Pointer to static configuration.
- Data  Pointer to received data.

**Parameters (inout):** State  Pointer to port/data communication state.

**Parameters (out):** None

**Return value:**

- Std_ReturnType E2E_E_INPUTERR_NULL
- E2E_E_INPUTERR_WRONG
- E2E_E_INTERR
- E2E_E_OK

For definitions for return values, see SWS_E2E_00047.

**Description:**

Checks the Data received using the E2E profile 1. This includes CRC calculation, handling of Counter and Data ID.

[SRS_E2E_08528, SRS_E2E_08527]

8.3.1.4 E2E_P01CheckInit

[SWS_E2E_00390] [ ]

**Service name:** E2E_P01CheckInit

**Syntax:**

```c
Std_ReturnType E2E_P01CheckInit(
    E2E_P01CheckStateType* StatePtr
)
```

**Service ID[hex]:** 0x1b

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):** None

**Parameters (inout):** StatePtr  Pointer to port/data communication state.

**Return value:**

- Std_ReturnType E2E_E_INPUTERR_NULL
- E2E_E_OK

For definitions for return values, see SWS_E2E_00047.

**Description:**

Initializes the protection state.
Specification of SW-C End-to-End Communication Protection Library
AUTOSAR CP Release 4.3.1

**Syntax:**
```
Std_ReturnType E2E_P01CheckInit(
    E2E_P01CheckStateType* StatePtr
)
```

**Service ID[hex]:** 0x1c

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):** None

**Parameters (inout):** None

**Parameters (out):** StatePtr Pointer to port/data communication state.

**Return value:**
- Std_ReturnType E2E_E_INPUTERR_NULL - null pointer passed
- E2E_E_OK

**Description:**
Initializes the check state

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

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

(SRS_E2E_08528, SRS_E2E_08527)

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

### 8.3.1.5 E2E_P01MapStatusToSM

**Parameters (in):**
- CheckReturn Return value of the E2E_P01Check function
- Status Status determined by E2E_P01Check function
- profileBehavior FALSE: check has the legacy behavior, before R4.2 TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2

**Parameters (out):** None

**Return value:** E2E_PCheckStatusType Profile-independent status of the reception on one single Data in one cycle.

[SWS_E2E_00382]
The function maps the check status of Profile 1 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 1 delivers a more fine-granular status, but this is not relevant for the E2E state machine.

This represents the R4.2 behavior:

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

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P01STATUS_OK</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P01STATUS_OKSOMELOST</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P01STATUS_SYNC</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P01STATUS_WRONGCRC</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P01STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P01STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P01STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
</tbody>
</table>

This represents the pre-R4.2 behavior:

[SWS_E2E_00476] If CheckReturn == E2E_E_OK and ProfileBehavior == FALSE, then the function E2E_P01MapStatusToSM shall return the values depending on the value of Status:

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P01STATUS_OK</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P01STATUS_OKSOMELOST</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P01STATUS_SYNC</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P01STATUS_WRONGCRC</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P01STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P01STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P01STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
</tbody>
</table>

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

8.3.2 E2E Profile 2 routines

8.3.2.1 E2E_P02Protect

[SWS_E2E_00160] Service name: E2E_P02Protect

Syntax:

```
Std_ReturnType E2E_P02Protect(
    const E2E_P02ConfigType* ConfigPtr,
    E2E_P02ProtectStateType* StatePtr,
    uint8* DataPtr
)
```
### 8.3.2.2 E2E_P02ProtectInit

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

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

### 8.3.2.3 E2E_P02Check

#### Service ID [hex]: 0x04
- **Sync/Async:** Synchronous
- **Reentrancy:** Reentrant
- **Parameters (in):**
  - ConfigPtr: Pointer to static configuration.
- **Parameters (inout):**
  - StatePtr: Pointer to port/data communication state.
- **Parameters (out):**
  - DataPtr: Pointer to the data to be protected.
- **Return value:**
  - Std_ReturnType
  - E2E_E_INPUTERR_NULL - null pointer passed
  - E2E_E_OK
- **Description:**
  - Checks the input parameters and initializes the protection state.
## 8.3.2.4 E2E_P02CheckInit

**Service name:** E2E_P02CheckInit

**Syntax:**
```
Std_ReturnType E2E_P02CheckInit(
   E2E_P02CheckStateType* StatePtr
)
```

**Service ID[hex]:** 0x1f

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):** None

**Parameters (inout):** StatePtr Pointer to port/data communication state.

**Return value:**
- Std_ReturnType
  - E2E_E_INPUTERR_NULL
  - E2E_E_INPUTERR_WRONG
  - E2E_E_INTERR
  - E2E_E_OK
  - For definitions for return values, see SWS_E2E_00047.

**Description:**
Check the array/buffer using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.

\[ (\text{SRS}_E2E\_08528, \text{SRS}_E2E\_08527) \]

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

1. LastValidCounter = 0
2. MaxDeltaCounter = 0
3. WaitForFirstData = TRUE
4. NewDataAvailable = TRUE
5. LostData = 0
6. Status = E2E_P02STATUS_NONEWDATA
7. NoNewOrRepeatedDataCounter = 0
8. SyncCounter = 0

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

## 8.3.2.5 E2E_P02MapStatusToSM

**Service name:** E2E_P02MapStatusToSM
### Syntax:

```c
E2E_PCheckStatusType E2E_P02MapStatusToSM(
    Std_ReturnType CheckReturn,
    E2E_P02CheckStatusType Status,
    boolean profileBehavior
)
```

### Service ID[hex]:
0x20

### Sync/Async:
Synchronous

### Reentrancy:
Reentrant

### Parameters (in):

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckReturn</td>
<td>Return value of the E2E_P02Check function</td>
</tr>
<tr>
<td>Status</td>
<td>Status determined by E2E_P02Check function</td>
</tr>
<tr>
<td>profileBehavior</td>
<td>FALSE: check has the legacy behavior, before R4.2</td>
</tr>
<tr>
<td></td>
<td>TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2</td>
</tr>
</tbody>
</table>

### Parameters (out): None

### Return value:

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P02STATUS_OK</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P02STATUS_OKSOMELOST</td>
<td></td>
</tr>
<tr>
<td>E2E_P02STATUS_SYNC</td>
<td></td>
</tr>
<tr>
<td>E2E_P02STATUS_WRONGCRC</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P02STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P02STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P02STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
<tr>
<td>E2E_P02STATUS_INITIAL</td>
<td></td>
</tr>
</tbody>
</table>

[SRS_E2E_08528, SRS_E2E_08527]

This represents the R4.2 behavior:

**[SWS_E2E_00380]**

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

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P02STATUS_OK</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P02STATUS_OKSOMELOST</td>
<td></td>
</tr>
<tr>
<td>E2E_P02STATUS_SYNC</td>
<td></td>
</tr>
<tr>
<td>E2E_P02STATUS_WRONGCRC</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P02STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P02STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P02STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
<tr>
<td>E2E_P02STATUS_INITIAL</td>
<td></td>
</tr>
</tbody>
</table>

[SRS_E2E_08528]

This represents the pre-R4.2 behavior:

**[SWS_E2E_00477]**

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

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P02STATUS_OK</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P02STATUS_OKSOMELOST</td>
<td></td>
</tr>
<tr>
<td>E2E_P02STATUS_INITIAL</td>
<td></td>
</tr>
<tr>
<td>E2E_P02STATUS_WRONGCRC</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P02STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P02STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P02STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
<tr>
<td>E2E_P02STATUS_SYNC</td>
<td></td>
</tr>
</tbody>
</table>

[SWS_E2E_00381] If CheckReturn != E2E_E_OK, then the function E2E_P02MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status). [SRS_E2E_08528]
8.3.3 E2E Profile 4 routines

8.3.3.1 E2E_P04Protect

[SWS_E2E_00338]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P04Protect</th>
</tr>
</thead>
</table>
| Syntax:     | Std_ReturnType E2E_P04Protect(
|             |   const E2E_P04ConfigType* ConfigPtr,
|             |   E2E_P04ProtectStateType* StatePtr,
|             |   uint8* DataPtr,
|             |   uint16 Length
|             | ) |
| Service ID[hex]: | 0x21 |
| Sync/Async: | Synchronous |
| Reentrancy: | Reentrant |
| Parameters (in): | ConfigPtr Pointer to static configuration. |
|                | Length Length of the data in bytes. |
| Parameters (inout): | StatePtr Pointer to port/data communication state. |
| Parameters (out): | None |
| Return value: | Std_ReturnType E2E_E_INPUTERR_NULL |
|                | E2E_E_INPUTERR_WRONG |
|                | E2E_E_INTERR |
|                | E2E_E_OK |
| Description: | Protects the array/buffer to be transmitted using the E2E profile 4. This includes checksum calculation, handling of counter and Data ID. |

] (SRS_E2E_08539, SRS_E2E_08527)

8.3.3.2 E2E_P04ProtectInit

[SWS_E2E_00373]

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

] (SRS_E2E_08539, SRS_E2E_08527)

[SWS_E2E_00377] In case State is NULL, E2E_P04ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.] (SRS_E2E_08539)
8.3.3.3 E2E_P04Check

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

](SRS_E2E_08539, SRS_E2E_08527)

8.3.3.4 E2E_P04CheckInit

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

](SRS_E2E_08539, SRS_E2E_08527)

[SWS_E2E_00378] In case State is NULL, E2E_P04CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:
1. Counter to 0xFF'FF.
2. Status to E2E_P04STATUS_ERROR. \(\text{[SRS_E2E_08539]}\)

### 8.3.3.5 E2E_P04MapStatusToSM

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P04MapStatusToSM</th>
</tr>
</thead>
</table>
| Syntax:     | E2E_PCheckStatusType E2E_P04MapStatusToSM(  
|             | Std_ReturnType CheckReturn,  
|             | E2E_P04CheckStatusType Status  
|             ) |
| Service ID[hex]: | 0x25 |
| Sync/Async: | Synchronous |
| Reentrancy: | Reentrant |
| Parameters (in): | CheckReturn  
|                 | Return value of the E2E_P04Check function  
|                 | Status  
|                 | Status determined by E2E_P04Check function |
| Parameters (inout): | None |
| Parameters (out): | None |
| Return value: | E2E_PCheckStatusType  
|                | Profile-independent status of the reception on one single Data in one cycle. |
| Description: | The function maps the check status of Profile 4 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 4 delivers a more fine-granular status, but this is not relevant for the E2E state machine.  
| \(\text{[SRS_E2E_08539, SRS_E2E_08527]}\) |

\(\text{[SWS_E2E_00351]}\) If CheckReturn = E2E_E_OK, then the function E2E_P04MapStatusToSM shall return the values depending on the value of Status:

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P04STATUS_OK or E2E_P04STATUS_OKSOMELOST</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P04STATUS_ERROR</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P04STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P04STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P04STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
</tbody>
</table>

\(\text{[SRS_E2E_08539]}\)

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

### 8.3.4 E2E Profile 5 routines

#### 8.3.4.1 E2E_P05Protect

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P05Protect</th>
</tr>
</thead>
</table>
| Syntax:     | Std_ReturnType E2E_P05Protect(  
|             | const E2E_P05ConfigType* ConfigPtr,  
|             | E2E_P05ProtectStateType* StatePtr,  
|             | uint8* DataPtr,  
|             ) |
8.3.4.2 E2E_P05ProtectInit

[SWS_E2E_00447]

| Service name: | E2E_P05ProtectInit |
| Service ID[hex]: | 0x27 |
| Sync/Async: | Synchronous |
| Reentrancy: | Reentrant |
| Parameters (in): | None |
| Parameters (inout): | None |
| Parameters (out): | StatePtr |
| Return value: | Std_ReturnType E2E_P05ProtectInit( |
| Description: | Initializes the protection state. |

In case State is NULL, E2E_P05ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.

[SWS_E2E_00448]

8.3.4.3 E2E_P05Check

[SWS_E2E_00449]

| Service name: | E2E_P05Check |
| Syntax: | Std_ReturnType E2E_P05Check( |

For definitions for return values, see SWS_E2E_00047.
### E2E_P05CheckInit

**Service ID [hex]:** 0x29  
**Sync/Async:** Synchronous  
**Reentrancy:** Reentrant  
**Parameters (in):** None  
**Parameters (inout):** None  
**Parameters (out):** StatePtr  
**Return value:** Std_ReturnType  
**Description:** Initializes the check state

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

1. Counter to 0xFF
2. Status to E2E_P05STATUS_ERROR.

### E2E_P05MapStatusToSM

**Service name:** E2E_P05MapStatusToSM  
**Syntax:**  
E2E_PCheckStatusType E2E_P05MapStatusToSM(  
  Std_ReturnType CheckReturn,
)

**Description:** Map the status to SM.
The function maps the check status of Profile 5 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 5 delivers a more fine-granular status, but this is not relevant for the E2E state machine.

8.3.5 E2E Profile 6 routines

8.3.5.1 E2E_P06Protect

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

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P05STATUS_OK or E2E_P05STATUS_OKSOMELOST</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P05STATUS_ERROR</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P05STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P05STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P05STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
</tbody>
</table>

If CheckReturn != E2E_E_OK, then the function E2E_P05MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).
8.3.5.2 E2E_P06ProtectInit

[SWS_E2E_00455] | 
---
**Service name:** E2E_P06ProtectInit

**Syntax:**

```c
Std_ReturnType E2E_P06ProtectInit(
    E2E_P06ProtectStateType* StatePtr
)
```

**Service ID[hex]:** 0x2c

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):** None

**Parameters (inout):** None

**Parameters (out):** StatePtr

**Return value:**

- Std_ReturnType E2E_E_INPUTERR_NULL - null pointer passed
- Std_ReturnType E2E_E_OK

**Description:**

Initializes the protection state.

[SRS_E2E_08539, SRS_E2E_08527]

In case State is NULL, E2E_P06ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.

8.3.5.3 E2E_P06Check

[SWS_E2E_00456] | 
---
**Service name:** E2E_P06Check

**Syntax:**

```c
Std_ReturnType E2E_P06Check(
    const E2E_P06ConfigType* ConfigPtr,
    E2E_P06CheckStateType* StatePtr,
    const uint8* DataPtr,
    uint16 Length
)
```

**Service ID[hex]:** 0x2d

**Sync/Async:** Synchronous

**Reentrancy:** Reentrant

**Parameters (in):**

- ConfigPtr: Pointer to static configuration.
- DataPtr: Pointer to received data.
- Length: Length of the data in bytes.

**Parameters (inout):** StatePtr: Pointer to port/data communication state.

**Parameters (out):** None

**Return value:**

- Std_ReturnType E2E_E_INPUTERR_NULL

**Description:**

For definitions for return values, see SWS_E2E_0047.
8.3.5.4 E2E_P06CheckInit

[SWS_E2E_00458]  

Service name: E2E_P06CheckInit  

Syntax:  

\[
\text{Std\_ReturnType } \text{E2E\_P06CheckInit}( \text{E2E\_P06CheckStateType\* StatePtr})
\]

Service ID[hex]: 0x2e  
Sync/Async: Synchronous  
Reentrancy: Reentrant  
Parameters (in): None  
Parameters (inout): None  
Parameters (out): StatePtr  

Description: Initializes the check state

In case State is NULL, E2E_P06CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:  
1. Counter to 0xFF  
2. Status to E2E_P06STATUS_ERROR. [SRS_E2E_08539, SRS_E2E_08527]

8.3.5.5 E2E_P06MapStatusToSM

[SWS_E2E_00459]  

Service name: E2E_P06MapStatusToSM  

Syntax:  

\[
\text{E2E\_P06MapStatusToSM}( \text{Std\_ReturnType CheckReturn,} \  
\text{E2E\_P06CheckStatusType Status})
\]

Service ID[hex]: 0x2f  
Sync/Async: Synchronous  
Reentrancy: Reentrant  
Parameters (in): CheckReturn, Status  
Parameters (inout): None  
Parameters (out): None  

Return value:  
- E2E_E_INPUTERR_NULL - null pointer passed  
- E2E_E_OK

Description: Returns the E2E status determined by E2E_P06Check function to the SM.
8.3.6 E2E Profile 7 routines

8.3.6.1 E2E_P07Protect

[SWS_E2E_00546] Service name: E2E_P07Protect
Syntax: Std_ReturnType E2E_P07Protect(
  const E2E_P07ConfigType* ConfigPtr,
  E2E_P07ProtectStateType* StatePtr,
  uint8* DataPtr,
  uint32 Length
) Service ID[hex]: 0x21
Sync/Async: Synchronous
Reentrancy: Reentrant

Parameters (in): ConfigPtr Pointer to static configuration.
Length Length of the data in bytes.

Parameters (inout): StatePtr Pointer to port/data communication state.
DataPtr Pointer to Data to be transmitted.

Parameters (out): None

Return value: Std_ReturnType E2E_E_INPUTERR_NULL
E2E_E_INPUTERR_WRONG
E2E_E_INTERR
E2E_E_OK

For definitions for return values, see SWS_E2E_00047.

Description: Protects the array/buffer to be transmitted using the E2E profile 7. This includes checksum calculation, handling of counter and Data ID.

[SRS_E2E_08539]
8.3.6.2 E2E_P07ProtectInit

[SWS_E2E_00547] [ ]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P07ProtectInit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Std_ReturnType E2E_P07ProtectInit(</td>
</tr>
<tr>
<td></td>
<td>E2E_P07ProtectStateType* StatePtr</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x22</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>StatePtr Pointer to port/data communication state.</td>
</tr>
<tr>
<td>Return value:</td>
<td>Std_ReturnType E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.</td>
</tr>
<tr>
<td>Description:</td>
<td>Initializes the protection state.</td>
</tr>
</tbody>
</table>

] (SRS_E2E_08539)

[SWS_E2E_00551] In case State is NULL, E2E_P07ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.} (SRS_E2E_08539)

8.3.6.3 E2E_P07Check

[SWS_E2E_00548] [ ]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P07Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Std_ReturnType E2E_P07Check(</td>
</tr>
<tr>
<td></td>
<td>const E2E_P07ConfigType* ConfigPtr,</td>
</tr>
<tr>
<td></td>
<td>E2E_P07CheckStateType* StatePtr,</td>
</tr>
<tr>
<td></td>
<td>const uint8* DataPtr,</td>
</tr>
<tr>
<td></td>
<td>uint32 Length</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x23</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>ConfigPtr Pointer to static configuration.</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>DataPtr Pointer to received data.</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Length Length of the data in bytes.</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>StatePtr Pointer to received data.</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>Std_ReturnType E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.</td>
</tr>
<tr>
<td>Description:</td>
<td>Checks the Data received using the E2E profile 7. This includes CRC calculation, handling of Counter and Data ID.</td>
</tr>
</tbody>
</table>

} (SRS_E2E_08539)
The function checks only one single data in one cycle, it does not
determine/compute the accumulated state of the communication link.

8.3.6.4 E2E_P07CheckInit

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

In case State is NULL, E2E_P07CheckInit shall return  
immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state  
structure, setting:  
  1. Counter to 0xFF'FF'FF'FF  
  2. Status to E2E_P07STATUS_ERROR.

8.3.6.5 E2E_P07MapStatusToSM

Service name: E2E_P07MapStatusToSM
Syntax: E2E_PCheckStatusType E2E_P07MapStatusToSM(  
  E2E_PCheckStatusType return,  
  E2E_P07CheckStatusType Status )
Service ID[hex]: 0x25
Sync/Async: Synchronous
Reentrancy: Reentrant
Parameters (in): return Profile-independent status of the reception on one  
  single Data in one cycle.  
  Status Status determined by E2E_P07Check function
Parameters (inout): None
Parameters (out): None
Return value: E2E_PCheckStatusType Profile-independent status of the reception on one  
  single Data in one cycle.
Description: The function maps the check status of Profile 7 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 7 delivers a more fine-granular status, but this is not relevant for the E2E state machine.

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

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P07STATUS_OK or E2E_P07STATUS_OKSOMELOST</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P07STATUS_ERROR</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P07STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P07STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P07STATUS_WrongSequence</td>
<td>E2E_P_WrongSequence</td>
</tr>
</tbody>
</table>

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

[SRS_E2E_08539]
8.3.7 E2E Profile 11 routines

8.3.7.1 E2E_P11Protect

[SWS_E2E_00575]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P11Protect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void E2E_P11Protect(</td>
</tr>
<tr>
<td></td>
<td>const E2E_P11ConfigType* ConfigPtr,</td>
</tr>
<tr>
<td></td>
<td>E2E_P11ProtectStateType StatePtr,</td>
</tr>
<tr>
<td></td>
<td>uint8 DataPtr,</td>
</tr>
<tr>
<td></td>
<td>uint16 Length</td>
</tr>
<tr>
<td></td>
<td>)</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x3b</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>ConfigPtr</td>
</tr>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>StatePtr</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>Protects the array/buffer to be transmitted using the E2E profile 11. This includes checksum calculation, handling of counter.</td>
</tr>
</tbody>
</table>

[SRS_E2E_08539, SRS_E2E_08527]

8.3.7.2 E2E_P11ProtectInit

[SWS_E2E_00576]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P11ProtectInit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void E2E_P11ProtectInit(</td>
</tr>
<tr>
<td></td>
<td>E2E_P11ProtectStateType* StatePtr</td>
</tr>
<tr>
<td></td>
<td>)</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x3c</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>StatePtr</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>Initializes the protection state.</td>
</tr>
</tbody>
</table>

[SRS_E2E_08539, SRS_E2E_08527]

[SWS_E2E_00555] In case State is NULL, E2E_P11ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0. [SRS_E2E_08539]

8.3.7.3 E2E_P11Check

[SWS_E2E_00572]

[SRS_E2E_08539]
### E2E_P11Check

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P11Check</th>
</tr>
</thead>
</table>
| Syntax:       | `void E2E_P11Check(
    const E2E_P11ConfigType* ConfigPtr,
    E2E_P11CheckStateType StatePtr,
    const uint8* DataPtr,
    uint16 Length
)` |
| Service ID[hex]: | 0x38 |
| Sync/Async: | Synchronous |
| Reentrancy: | Reentrant |
| Parameters (in): | ConfigPtr Pointer to static configuration. DataPtr Pointer to received data. Length Length of the data in bytes. |
| Parameters (inout): | StatePtr Pointer to port/data communication state. |
| Parameters (out): | None |
| Return value: | None |
| Description: | Checks the Data received using the E2E profile 11. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. |

---

8.3.7.4 E2E_P11CheckInit

[SWS_E2E_00573] [SWS_E2E_00556]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_P11CheckInit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td><code>void E2E_P11CheckInit(E2E_P11CheckStateType* StatePtr)</code></td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x39</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>StatePtr Pointer to port/data communication state.</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>Initializes the check state</td>
</tr>
</tbody>
</table>

---

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

1. Counter to 0xE
2. Status to E2E_P11STATUS_ERROR.]

---

8.3.7.5 E2E_P11MapStatusToSM

[SWS_E2E_00574]
Service name: E2E_P11MapStatusToSM

Syntax:

```c
void E2E_P11MapStatusToSM(
    Std_ReturnType CheckReturn,
    E2E_P11CheckStatusType Status
)
```

Service ID[hex]: 0x3a

Sync/Async: Synchronous

Reentrancy: Reentrant

Parameters (in): | Return value of the E2E_P11Check function
| Status determined by E2E_P11Check function

Parameters (inout): None

Parameters (out): None

Return value: None

Description:
The function maps the check status of Profile 11 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 11 delivers a more fine-granular status, but this is not relevant for the E2E state machine.

[SWS_E2E_00557] If CheckReturn = E2E_E_OK, then the function E2E_P05MapStatusToSM shall return the values depending on the value of Status:

<table>
<thead>
<tr>
<th>Status</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E_P11STATUS_OK or E2E_P11STATUS_OKSOMELOST</td>
<td>E2E_P_OK</td>
</tr>
<tr>
<td>E2E_P11STATUS_ERROR</td>
<td>E2E_P_ERROR</td>
</tr>
<tr>
<td>E2E_P11STATUS_REPEATED</td>
<td>E2E_P_REPEATED</td>
</tr>
<tr>
<td>E2E_P11STATUS_NONEWDATA</td>
<td>E2E_P_NONEWDATA</td>
</tr>
<tr>
<td>E2E_P11STATUS_WRONGSEQUENCE</td>
<td>E2E_P_WRONGSEQUENCE</td>
</tr>
</tbody>
</table>

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

[SRS_E2E_08539] (SRS_E2E_08539, SRS_E2E_08527)
8.3.8 E2E Profile 22 routines

8.3.8.1 E2E_P22Protect

| [SWS_E2E_00580] | 
|---|---|
| **Service name:** | E2E_P22Protect |
| **Syntax:** | void E2E_P22Protect(  
|  |  
|  | const E2E_P22ConfigType* ConfigPtr,  
|  | E2E_P22ProtectStateType StatePtr,  
|  | uint8 DataPtr,  
|  | uint16 Length  
|  | ) |
| **Service ID[hex]:** | 0x40 |
| **Sync/Async:** | Synchronous |
| **Reentrancy:** | Reentrant |
| **Parameters (in):** | ConfigPtr | Pointer to static configuration. |
|  | Length | Length of the data in bytes |
| **Parameters (inout):** | StatePtr | Pointer to port/data communication state. |
|  | DataPtr | Pointer to Data to be transmitted. |
| **Parameters (out):** | None |
| **Return value:** | None |
| **Description:** | Protects the array/buffer to be transmitted using the E2E profile 22. This includes checksum calculation, handling of counter. |

](SRS_E2E_08539, SRS_E2E_08527)

8.3.8.2 E2E_P22ProtectInit

| [SWS_E2E_00581] | 
|---|---|
| **Service name:** | E2E_P22ProtectInit |
| **Syntax:** | void E2E_P22ProtectInit(  
|  | E2E_P22ProtectStateType* StatePtr  
|  | ) |
| **Service ID[hex]:** | 0x41 |
| **Sync/Async:** | Synchronous |
| **Reentrancy:** | Reentrant |
| **Parameters (in):** | None |
| **Parameters (inout):** | None |
| **Parameters (out):** | StatePtr | Pointer to port/data communication state. |
| **Return value:** | None |
| **Description:** | Initializes the protection state. |

](SRS_E2E_08539, SRS_E2E_08527)

[SWS_E2E_00559] In case State is NULL, E2E_P22ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall intialize the state structure, setting Counter to 0.](SRS_E2E_08539)

8.3.8.3 E2E_P22Check

| [SWS_E2E_00577] | 
|---|---|
### Service name: E2E_P22Check

#### Syntax:
```c
void E2E_P22Check(
    const E2E_P22ConfigType* ConfigPtr,
    E2E_P22CheckStateType StatePtr,
    const uint8* DataPtr,
    uint16 Length
)
```

#### Service ID[hex]: 0x3d

#### Sync/Async: Synchronous

#### Reentrancy: Reentrant

#### Parameters (in):
- **ConfigPtr**: Pointer to static configuration.
- **DataPtr**: Pointer to received data.
- **Length**: Length of the data in bytes.

#### Parameters (inout):
- **StatePtr**: Pointer to port/data communication state.

#### Parameters (out): None

#### Return value: None

#### Description:
Checks the Data received using the E2E profile 22. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.

---

### Service name: E2E_P22CheckInit

#### Syntax:
```c
void E2E_P22CheckInit(
    E2E_P22CheckStateType* StatePtr
)
```

#### Service ID[hex]: 0x3e

#### Sync/Async: Synchronous

#### Reentrancy: Reentrant

#### Parameters (in): None

#### Parameters (inout): None

#### Parameters (out):
- **StatePtr**: Pointer to port/data communication state.

#### Return value: None

#### Description:
Initializes the check state.

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

3. Counter to 0xF
4. Status to E2E_P22STATUS_ERROR.

---

### 8.3.8.5 E2E_P22MapStatusToSM

#### Description:
Initializes the check state.
Service name: | E2E_P22MapStatusToSM  
|---|---  
Syntax: | void E2E_P22MapStatusToSM(  
| | Std_ReturnType CheckReturn,  
| | E2E_P22CheckStatusType Status  
)  
Service ID[hex]: | 0x3f  
Sync/Async: | Synchronous  
Reentrancy: | Reentrant  
Parameters (in): | CheckReturn | Return value of the E2E_P22Check function  
| Status | Status determined by E2E_P22Check function  
Parameters (inout): | None  
Parameters (out): | None  
Return value: | None  
Description: | The function maps the check status of Profile 22 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 22 delivers a more fine-granular status, but this is not relevant for the E2E state machine.  
(SRS_E2E_08539, SRS_E2E_08527)  

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

| Status | Return value  
|---|---  
| E2E_P22STATUS_OK or E2E_P22STATUS_OKSOMELOST | E2E_P_OK  
| E2E_P22STATUS_ERROR | E2E_P_ERROR  
| E2E_P22STATUS_REPEATED | E2E_P_REPEATED  
| E2E_P22STATUS_NONEDATA | E2E_P_NONEDATA  
| E2E_P22STATUS_WRONGSEQUENCE | E2E_P_WRONGSEQUENCE  
(SRS_E2E_08539)  

[SWS_E2E_00562] If CheckReturn != E2E_E_OK, then the function E2E_P22MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).  
(SRS_E2E_08539)
8.3.9 E2E State machine routines

8.3.9.1 E2E_SMCheck

[SWS_E2E_00340] [E2E_SMCheck]

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

] (SRS_E2E_08539)

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

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

8.3.9.2 E2E_SMCheckInit

[SWS_E2E_00353] [E2E_SMCheckInit]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2E_SMCheckInit</th>
</tr>
</thead>
</table>
| Syntax:     | Std_ReturnType E2E_SMCheckInit(   
|             |   E2E_SMCheckStateType* StatePtr,   
|             |   const E2E_SMConfigType* ConfigPtr   
|             | )                                     |
| Service ID[hex]: | 0x31 |
| Sync/Async: | Synchronous |
| Reentrancy: | Reentrant |
| Parameters (in): | ConfigPtr Pointer to configuration of the state machine |
| Parameters (in/out): | None |
| Parameters (out): | StatePtr Pointer to port/data communication state. |

- AUTOSAR confidential -
Return value: 
| Std_ReturnType | E2E_E_INPUTERR_NULL - null pointer passed | E2E_E_OK |

Description: 
- Initializes the state machine.

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

Else (i.e. both pointers are not NULL), the function E2E_SMCheckInit shall initialize the State structure, setting:
1. ProfileStatusWindow[] to E2E_P_NOTAVAILABLE on each element of the array
2. WindowTopIndex to 0
3. OKCount to 0
4. ERRORCount to 0
5. SMState to E2E_SM_NODATA

and it shall return with E2E_E_OK. [SRS_E2E_08539]

8.3.10 Auxiliary Functions

8.3.10.1 E2E_GetVersionInfo

[SWS_E2E_00032] [SWS_E2E_00033] The function E2E_GetVersionInfo shall return the version information of this module. The version information includes:
- vendor ID
- module ID
- sw_major_version
- sw_minor_version
- sw_patch_version

[SRS_E2E_08528]
8.4 Call-back notifications

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

8.5 Scheduled functions

None. The E2E library does not have scheduled functions.

8.6 Expected Interfaces

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

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

8.6.1 Mandatory Interfaces

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

Figure 8-22: Expected mandatory interfaces by E2E library
9 Sequence Diagrams for invoking E2E Library

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

9.1 Sender

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

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

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

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

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

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

9.1.1 Sender of data elements

The diagram below specifies the overall sequence involving the E2E Library called by the Sender of data elements. The Sender itself can be realized by one or more modules/files. After the diagram, there are requirements specific to Sender of data elements.
Figure 9-1: Sender of data elements

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

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

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

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

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

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

9.1.2 Sender at signal group level

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

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

Note: When selecting the following initialization and configuration parameters the functional behaviour of the enhanced E2E_PxxCheck()-functions (introduced in AUTOSAR R4.0.4 and R3.2.2) is application-wise backward compatible to the E2E_PxxCheck()-function of the earlier AUTOSAR releases:
State → SyncCounter := 0;
Config → MaxNoNewOrRepeatedData := 14 (when using Profile 1);
Config → MaxNoNewOrRepeatedData := 15 (when using Profile 2);
Config → SyncCounterInit := 0;

Exemplary configuration parameters and resulting behaviour of the E2E_PxxCheck function:
E2E_PxxConfigType:
- Config → MaxDeltaCounterInit = 2 (i.e. tolerance interval for initial counter differences)
- Config → MaxNoNewOrRepeatedData = 3 (i.e. tolerance interval for maximum counter differences)
- Config → SyncCounterInit = 2 (i.e. duration of counter continuity check)
Timeout interval checked by SWC = 8 transmission cycles

Exemplary configuration parameters of the E2E_PxxCheck() function and their effects

Figure 9-3: Configuration parameters of the E2E_PxxCheck() function and their effects
Clarification regarding SYNC states in Figure 9-3: In cycle 9, the counter value is not trustable anymore since the NoNewOrRepeatedData exceeds MaxNoNewOrRepeatedData. The resulting behavior is similar to as if an "unexpected behavior of the counter" is detected in cycle 9. Thus, the "counter continuity check" spans from cycle 10-11.

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

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

[UC_E2E_00208] During its initialization, the Receiver shall initialize the E2E_PXXCheckStateType with the following values:
LastValidCounter = 0
MaxDeltaCounter = 0
SyncCounter = 0
NoNewOrRepeatedDataCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA] (SRS_E2E_08528)

[UC_E2E_00209] In every receive cycle, the Receiver shall:
1. Invoke once the reception function Rte_Read_<p>_<o>().
2. Set the attribute State->NewDataAvailable to TRUE if new data has been received without any errors:
   a. In case of single channel or channel 1: State->NewDataAvailable = (retRteRead == RTE_E_OK) ? TRUE : FALSE;
   b. In case of channel 2: State->NewDataAvailable = TRUE; (note: the second channel has no access to Rte_Read return value).
3. Update Data, using received data element or I-PDU.
4. Call once the function E2E_PXXCheck().
5. Handle results (return value and State parameter) returned by E2E_PXXCheck().] (SRS_E2E_08528)

Note: In case of single channel only, the NewDataAvailable flag may additionally incorporate the return value of the Rte_IsUpdated() API (if available) in the following way:
1. Invoke once the function Rte_IsUpdated_<p>_<o>().
2. Distinguish
   a) If Rte_IsUpdated_<p>_<o>() returned FALSE : Set the attribute State->NewDataAvailable to FALSE and retRteRead to RTE_E_OK
   b) If Rte_IsUpdated_<p>_<o>() returned TRUE :
      i. Invoke once the reception function Rte_Read_<p>_<o>()
      ii. Set the attribute State->NewDataAvailable to TRUE if Rte_Read_<p>_<o>() returned RTE_E_OK, otherwise set it to FALSE
3. Steps 3.-5. as stated in [UC_E2E_00209].

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

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

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

9.2.1 Receiver at data element level

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

**Figure 9-4: Receiver of data elements**

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

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

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

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

9.2.2 Receiver at signal group level

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

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

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

- AUTOSAR confidential -
Figure 9-5: Receiver of I-PDUs
10 Configuration specification

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

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

10.1 Published Information

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

Additional module-specific published parameters are listed below if applicable.
11 Annex A: Safety Manual for usage of E2E Library

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

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

11.1 E2E profiles and their standard variants

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

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

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

11.2 E2E error handling

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

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

[UC_E2E_00235] The user (caller) of E2E Library, in particular the receiver, shall provide the error handling mechanisms for the faults detected by the E2E Library.] (SRS_E2E_08528)

11.3 Maximal lengths of Data, communication buses

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

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

<table>
<thead>
<tr>
<th>E2E Profile</th>
<th>Max applicable length including control fields for inter-ECU communication</th>
</tr>
</thead>
</table>

(continued)
In E2E Profiles 1 and 2, the Hamming Distance is 2, up to the given lengths. Due to 8 bit CRC, the burst error detection is up to 8 bits.

[UC_E2E_00051] In case of inter-ECU communication over FlexRay, the length of the complete Data (including application data, CRC and counter) protected by E2E Profile 1 or E2E Profile 2 should not exceed 32 bytes.] (SRS_E2E_08528)

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

[UC_E2E_00061] In case of CAN or LIN the length of the complete data element (including application data, CRC and counter) protected by E2E Profile 1 should not exceed 8 bytes.] (SRS_E2E_08528)

[UC_E2E_00315] In case of inter-ECU, the length of the complete Data (including application data and E2E header) protected by E2E Profile 4, 5 or 6 shall not exceed 4kB.] (SRS_E2E_08528)

The requirements UC_E2E_00051, UC_E2E_00061 and UC_E2E_00315 only contain a reasonable maximum length evaluated during the design of the E2E profiles.

[UC_E2E_00236] When using E2E Library, the designer of the functional or technical safety concept of a particular system using E2E Library shall evaluate the maximum permitted length of the protected Data in that system, to ensure an appropriate error detection capability.] (SRS_E2E_08539)

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

If the protected data length exceeds the network bus frame limit (or payload limit), the data can be segmented on the sender side after the E2E protection, and be assembled on the receiver side before the E2E evaluation. The possible faults happening during segmentation/desegmentation can be considered as "corruption of information".
When designing the functional or technical safety concept of a particular system any user of E2E Library shall ensure that the transmission of one undetected erroneous data element in a sequence of data elements between sender and receiver will not directly lead to the violation of a safety goal of this system.

In other words, SW-C shall be able to tolerate the reception of one erroneous data element, which error was not detected by the E2E library. What is not required is that an SW-C tolerates two consecutive undetected erroneous data elements, because it is enough unlikely that two consecutive Data are wrong AND that for both Data the error remains undetected by the E2E library. \( (SRS\text{\_E2E\_08528, SRS\_E2E\_08537}) \)

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

11.4 Methodology of usage of E2E Library

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

In the first step, the user selects the architectural approach how E2E Library is used in a given system (through COM callouts, through E2E Protection wrapper etc). There are several architectural solutions of usage of E2E Library described in Chapter 11.9.

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

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

1. For each signal group to be protected, there is a separate instance of EndToEndDescription, associated in System Template to ISignalIPdu metaclass.

2. For each data element to be protected, there is a separate instance of EndToEndDescription, associated indirectly to VariableDataPrototype, SenderComSpec and ReceiverComSpec metaclasses.
In the fourth and last step, the user generates (or otherwise develops) the necessary glue code (e.g. E2E Protection Wrapper, COM callouts), responsible for invocation of E2E Library functions. The glue code serves as an adapter between the communication modules (e.g. COM, RTE) and E2E Library.

11.5 Configuration constraints on Data IDs

11.5.1 Data IDs

To be able to verify the identity of the data elements or signal groups, none of two are allowed to have the same Data ID (E2E Profiles 1, 4, 5, 6, 7) or same DataIDList[] (E2E Profile 2) within one system of communicating ECUs.

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

[UC_E2E_00071] Any user of E2E Library shall ensure that within one implementation of a communication network every safety-related data element, protected by E2E Library, has a unique Data ID (E2E Profiles 1, 4, 5, 6, 7) or a unique DataIDList[] (for E2e Profile 2).] (SRS_E2E_08528)

[UC_E2E_00237] Any user of E2E Library shall ensure, that within one implementation of a communication network every safety-related Data, protected by E2E Library, has a unique Data ID (E2E Profiles 1, 4, 5, 6, 7) or a unique DataIDList[].] (SRS_E2E_08528)

Note: For Profile 1 requirement (UC_E2E_00071) may not be sufficient in some cases, because Data ID is longer than CRC, which results with additional requirements UC_E2E_00072 and UC_E2E_00073. In Case of Profile 1 the ID can be encoded in CRC by double Data ID configuration (both bytes of Data ID are included in CRC every time), or in alternating Data ID configuration (high byte or low byte of Data ID are put in CRC alternatively, depending of parity of Counter), there are different additional requirements/constraints described in the sections below.

11.5.2 Double Data ID configuration of E2E Profile 1

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

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

```plaintext
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI1), highbyte(DI1)} )
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI2), highbyte(DI2)} )
```

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

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

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

```plaintext
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI1), highbyte(DI1)} )
Crc_CalculateCRC8( start value: 0x00, data[2]: {lowbyte (DI2), highbyte(DI2)} )
```

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

**11.5.3 Alternating Data ID configuration of E2E Profile 1**

In the alternating Data ID configuration, either high byte or low byte of Data ID is put in CRC alternatively, depending of parity of Counter. In this configuration, two consecutive Data are needed to verify the data identity. This is not about the reliability of the checksum or software, but really the algorithm constraint, as on every single Data only a single byte of the Data ID is transmitted and therefore it requires two consecutive receptions to verify the Data ID of received Data.
11.5.4 Nibble configuration of E2E Profile 1

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

[UC_E2E_00308] Any user of Profile 1 in Nibble Data ID configuration shall ensure that:

1. the high nibble of high byte of Data ID is equal to 0
2. the low nibble of high byte of Data ID is within the range 0x1..0xE (to avoid collisions with other E2E Profile 1 configurations that have 0x0 on this nibble, and to exclude the invalid value 0xF).
3. The low byte of Data ID is different to low byte of any Data ID present in the same bus that uses E2E Profile in Double Data ID configuration.

(SRS_E2E_08528)

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

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

(SRS_E2E_08527)

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

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

11.6 Building custom E2E protocols

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

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

(SRS_E2E_08528)
A list of CRC routines is provided by E2E Library. CRC should be calculated on the bytes and bits of the data elements in the same order as in which it is transmitted on hardware bus. To be able to do this, the microcontroller Endianness and the used bus must be known. Once it is known, the corresponding E2E Library CRC routines should be used.

11.7 I-PDU Layout

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

11.7.1 Alignment of signals to byte limits

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

[UC_E2E_00062] When using E2E Profiles, signals that have length < 8 bits should be allocated to one byte of an I-PDU, i.e. they should not span over two bytes. (SRS_E2E_08528)

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

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

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

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

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

Figure 11-2: Example for alignment not following recommendations

11.7.2 Unused bits

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

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

(SRS_E2E_08528)

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

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

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

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

(SRS_E2E_08528)

11.7.3 Byte order (Endianness)

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

1. start with the least significant byte first – the significance of the byte increases with the increasing byte significance. This is called littleEndian (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 bigEndian (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:

<table>
<thead>
<tr>
<th>Network</th>
<th>Byte order</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlexRay</td>
<td>Little Endian</td>
</tr>
<tr>
<td>CAN</td>
<td>Little Endian</td>
</tr>
<tr>
<td>LIN</td>
<td>Little Endian</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Big Endian</td>
</tr>
<tr>
<td>Byteflight (not supported by AUTOSAR)</td>
<td>Big Endian</td>
</tr>
<tr>
<td>MOST (not supported by AUTOSAR)</td>
<td>Big Endian</td>
</tr>
</tbody>
</table>

Table 11-1: Networks and their byte order

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

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

However, the TCP/IP stack is Big Endian. The E2E Profile 4, 6 and 7 can be used for FlexRay TP and CAN TP, but the main use case is TCP/IP. Moreover, TCP/IP can be considered as more future oriented, therefore Big Endian is foreseen for E2E Profile 4 and 6:

**[UC_E2E_00316]** Any user of E2E Profile 4, 6 and 7 shall place multibyte data in Big Endian order.] (SRS_E2E_08539)

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

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

For example, a signal/data element as non-opaque on sender side and opaque on receiver side or vice versa are not allowed. [SRS_E2E_08528]

11.7.4 Bit order

There are two typical ways to store the bits of a byte:
1. most significant bit first (MSB first)
2. or least significant bit first (LSB first).

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

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

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

<table>
<thead>
<tr>
<th>Network</th>
<th>Bit order</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlexRay</td>
<td>MSB first</td>
</tr>
<tr>
<td>CAN</td>
<td>MSB first</td>
</tr>
<tr>
<td>LIN</td>
<td>LSB first</td>
</tr>
<tr>
<td>Ethernet</td>
<td>LSB first</td>
</tr>
<tr>
<td>Byteflight (not supported by AUTOSAR up to Release 4.0)</td>
<td>MSB first</td>
</tr>
<tr>
<td>MOST (not supported by AUTOSAR up to Release 4.0)</td>
<td>MSB first</td>
</tr>
</tbody>
</table>

Table 11-2: Networks and their bit order

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

The bit order of the microcontroller is independent from the bit order of the network, but in all cases (combinations of different bit endianness of network sender and receiver microcontrollers) there is no impact on the user of E2E due to bit order.
11.8 RTE configuration constraints for SW-C level protection

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

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

11.8.1 Communication model for SW-C level protection

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

[UC_E2E_00087] In case the E2E Library is used to protect data elements, then the user of E2E Library shall use the Sender-receiver communication model for safety-related communication. (SRS_E2E_08528)

11.8.2 Multiplicities for SW-C level protection

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

[UC_E2E_00258] In case the E2E Library is used to protect data elements, then the selected multiplicity shall be 1:N or 1:1. (SRS_E2E_08528)

11.8.3 Explicit access

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

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

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

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

As explained in 7.3.5, E2E Profile 1 and 2 together with the proposed E2E protection wrapper provide timeout detection (which is one of the failure modes to handle – e.g. message loss). This is achieved by having the receiver executing independently from the reception of the data, and by the usage of a counter within E2E Profiles. By this means, if e.g. a data element is lost, it is seen by the receiver that every time the
read data element has the same counter. This however requires that the receiver is not solely executed upon the arrival of data.

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

[UC_E2E_00089] In case the E2E Library is used to protect data elements, data elements accessed with E2E Protection Wrapper shall use the activation “Explicit data read access” (i.e. it shall not use the activations ”Implicit data read access”).] (SRS_E2E_08528)

11.9 Restrictions on the use of COM features

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

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

The restriction classes are as follows:

- "supported" means that both (E2E COM Callout and E2EPW) do support this feature.
- "use case dependent" means that the feature might be used/usable depending on the actual use case and configuration on sender and receiver side. However, suitability for an actual system and its influence on the safety requirements has to be analysed.
- "not supported" means that at least one variant (either E2E COM Callout or E2EPW) does not support this feature or a failure mode can be masked.
<table>
<thead>
<tr>
<th>COM Feature / brief description</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SRS_Com_02078] Support of endianess conversion</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_02086] Support of Sign-Extension for received signals</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_02042] Initialization of unused areas/ bits of an I-PDU</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_02083] Transmission Modes</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02082] Two different Transmission Modes</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02084] Signal data based selection of Transmission Mode</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02113] Signal data based transmission modes for configured serialized data</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02046] Configuration of signal notification</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_02089] Timeout indication mechanism on receiver-side</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_02080] Value substitution in case of a signal timeout</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02089] Cancelation outstanding repetitions in case of a new send request</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02077] Signal invalidation mechanism on sender-side</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02079] Signal invalidation mechanism on receiver-side</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02087] Substitution of invalid value by configurable data value</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02088] Substitution of the last received value by the init value in case of signal timeout</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_00218] Starting/ Stopping communication of I-PDU groups</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_00192] Enabling/ disabling reception deadline monitoring of I-PDU groups</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02041] Consistent transfer of complex data types</td>
<td>supported</td>
</tr>
<tr>
<td>[SRS_Com_02091] Placement of large or dynamical length signals</td>
<td>not supported</td>
</tr>
<tr>
<td>[SRS_Com_02092] Support only one dynamic length signal per I-PDU</td>
<td>not supported</td>
</tr>
<tr>
<td>[SRS_Com_02093] Dynamic length signal must be placed last in I PDU</td>
<td>not supported</td>
</tr>
<tr>
<td>[SRS_Com_02094] Dynamic length signals must be of type UINT8[n]</td>
<td>not supported</td>
</tr>
<tr>
<td>[SRS_Com_02095] TP shall be used to fragment and reassemble large signals and dynamical signals</td>
<td>not supported</td>
</tr>
<tr>
<td>[SRS_Com_02030] Identify if a signal/signal group is updated by the sender</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02058] Deadline monitoring of receiving updated signals/signal groups</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02099] I-PDU Counter mechanism</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02100] I-PDU Counter configuration</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02101] Transmission and reception using I-PDU Counter</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02102] I-PDU Counter error handling</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02103] I-PDU Replication mechanism</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02104] I-PDU replication configuration</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02105] Transmission and reception using I-PDU Replication</td>
<td>use case dependent</td>
</tr>
<tr>
<td>[SRS_Com_02106] I-PDU Replication error handling</td>
<td>use case dependent</td>
</tr>
<tr>
<td>Minimum Delay Time</td>
<td>use case dependent</td>
</tr>
</tbody>
</table>
### 11.10 Examples for the implementation of E2E protection concepts based on E2E-Library - Branch

Note: this has been moved from chapter 12.

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

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

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

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

Note: For an E2E protection concept that focuses on ensuring the availability of data an implementation of the communication channel, with a sufficient fault tolerance is needed (e.g. using independent redundant channels). The usage of redundant communication channels may create a need for additional safety mechanisms e.g. to ensure the consistency of the data streams when transmitted independently.

#### 11.10.1 Basic principles

Typical basic principles for effective E2E protection concepts are:

- In normal operation mode, the sender ensures that it sends out valid data on a regular basis (e.g. cyclic).
- In this context valid data can be:
  - Data fully complying with their required safety-related properties;
  - Data complying with their required safety-related properties to the extent signaled by an additionally provided qualifier (i.e. signal qualifier);
  - Data explicitly labeled as invalid data (e.g. using an signal invalid value)
- In normal operation mode, the sender groups the data as pre-determined (e.g. to ensure consistency for a set of data) and protects the grouped data with suitable protection mechanisms (e.g. by using the protect functions provided by E2E-Library) prior to their transmission.
- In case of an internal fault, the sender ensures that it sends out either data explicitly labeled as invalid (i.e. only the specific data elements that are
possibly affected by this internal fault) or else no data (i.e. fail-safe respective fail-silent behavior of sender in case of a severe fault).

- The infrastructure used for data transmission from a sender to the receiver(s) (e.g. BSWM, Buses, Gateways, etc.) is designed and implemented in such way that it cannot systematically interfere with the used E2E-protection (e.g. by unpacking protected data including the re-calculation of their CRC).

- In normal operation mode, the receiver monitors whether new data has arrived on a regular basis (e.g. cyclic) independently from an external trigger condition coming from elements to which it wants to achieve freedom from interference (e.g. COM).

- In normal operation mode, the receiver is able to detect relevant communication faults within its determined time interval by evaluating the protection mechanisms of the received data and its internal timeout monitoring.

- In case of an detected communication fault, the receiver autonomously realizes the necessary reactions to mitigate the detected communication fault within its determined time interval in compliance with the functional safety concept of the system (i.e. fail-safe respective fail-silent behavior of receiver)

- The fault tolerance time interval of the respective safety-related system is not violated when adding up the allowed time interval for the detection and mitigation of faults at the sender, the time interval required for robustness of data transmission during normal operation (e.g. to compensate gateways) and the allowed time interval for the detection and mitigation of faults at the receiver.

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

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

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

- evaluate each received protected data (e.g. by using the check functions provided by E2E-Library)

- evaluate all protected data it received within its determined time interval for error detection and qualification $t_{EDO}$ up to the data it received at last.

To evaluate both aspects for the determination of communication integrity a receiver can implement a monitoring function as shown in Figure 11-3:
Figure 11-3: Example for a monitoring function to determine the integrity of communication within a receiver

To implement this monitoring function the receiver creates a history of the data it received. Received valid data (i.e. status of check function is e.g. E2EPW_STATUS_OK or E2EPW_STATUS_OKSOMELOST) is stored with a history as follows:

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

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

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

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

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

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

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

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

The fault tolerance designed into the receiver (see UC_E2E_00170) may be exceeded as a possible consequence.
12 Annex B: Application hints on usage of E2E Library

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

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

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

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

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

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

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

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

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

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

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

All necessary options, enabling to generate the code for the described solutions are available in AUTOSAR configuration, defined in System Template [12] and Software Component Template [11]. This contains e.g. association of I-PDUs with Data IDs.
To generate the wrapper, the user defines EndToEnd* metaclasses and associates them to VariableDataPrototypes (representing complex data elements). To generate the COM E2E callouts for an I-PDU, the user defines EndToEnd* metaclasses and associates them to ISignalIPdu metaclass (representing the I-PDU).

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

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

### 12.1 E2E Protection Wrapper

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

The usage of E2E Protection Wrapper allows the use of VFB communication between SW-Cs\(^1\), without the need of further measures to ensure VFB's integrity.

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

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

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

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

---

\(^1\) The term SW-C includes any software module that has an RTE interface, i.e. a sensor/actuator/application SW-C, an AUTOSAR service, or a Complex Driver.
[UC_E2E_00292] If the E2E Library is invoked from E2E Protection Wrapper (at the level of data elements), then multiple instantiation is not allowed. For an AUTOSAR software component which uses the E2E Protection Wrapper the value of the attribute supportsMultipleInstantiation of the SwcInternalBehavior shall be set to FALSE in the AUTOSAR software component description. The E2E Protection Wrapper itself is not a part of E2E Library. However, its options are standardized. Most of the options for E2E Protection Wrapper are in System Template [12] and some of them are in Software Component Template [11].] (SRS_E2E_08528)

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

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

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

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

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

12.1.1 Functional overview

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

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

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

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

Figure 12-1 shows the overall flow of usage of E2E Library and E2E Protection Wrapper from SW-Cs (the 1st number on the labels defines the order of execution):
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
In this example, for SW-C1 and SW-C2 it is not visible that the communication is going through such a Transmission Manager, which can support the portability and optimize resource usage of communication network. It is only through AUTOSAR configuration where it is visible that the receiver of SW-C1 and of SW-C2 is SW-C3.

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

12.1.3 Application scenario with E2E Manager and Conversion Manager

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

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

The E2E Manager and Conversion Manager SW-C modules are not part of E2E Library nor part of AUTOSAR.
Figure 12-5: E2E Manager and Conversion Manager – sender ECU

Figure 12-6: E2E Manager and Conversion Manager – receiver ECU
In the above example, the SW-Cs of sender ECU generate three data elements (dataEl1, dataEl2 and dataEl3) but the SW-Cs of receiver ECU use only two data elements (dataEl1 and dataEl2). The unused DataEl3c is not delivered to Conversion Manager. Thanks to this, if due to e.g. system evolution, the definition of DataEl3 changes, then the receiver SW-Cs (SW-C 5, SW-C 6 and SW-C 7 Conversion Manager) do not need to be changed.

The corresponding system configuration description looks as shown by Figure 12-7. Note that the SW-C 7 has as input only the required data elements. The unused data elements (CRC, counter, dataEl3c) are not provided:

![Figure 12-7: E2E Manager and Conversion Manager - system configuration](image)

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

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

```c
/* the input complex data element contains primitive data elements unused by other SW-Cs of the ECU */
typedef struct {
    uint8 crc;
    uint8 counter;
    uint16 dataEl1c;
    uint16 dataEl2c;
    uint16 dataEl3c;
} Inputswc8Type;

/* the output complex data element is a subset of input, with the data used by other SW-Cs of the ECU */
typedef struct {
    uint16 dataEl1c;
    uint16 dataEl2c;
} Outputswc8DataType;

Inputswc8Type Inputswc8;
Outputswc8Type Outputswc8;
```
/* copy from Inputswc8 the primitive data elements that are also in outputswc8 */

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

[UC_E2E_00274] E2E Manager shall have complex data elements with prefix Input or with prefix Output. There is one-to-one relationship between the data element with input prefix and data element with output prefix. (SRS_E2E_08528)

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

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

In the example above, Outputswc8 contains the subset of attributes of Inputswc8. It does not contain dataEl3c, crc, nor counter.

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

In the example above, the application code of E2E manager copies dataEl1c and dataEl2c from Inputswc8 to Outputswc8.

[UC_E2E_00272] The implementation of the Conversion Manager and E2E Manager (as a safety-related Software Component), shall comply with the requirements for the development of safety-related software for automotive domain. (SRS_E2E_08528)

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

12.1.4 File structure

The figure below shows the required file structure of E2E Protection Wrapper.
The E2E Protection Wrapper interface generated by an application generator.

E2EPW_<SWC-Type-short name>.c

E2EPW_<SWC-Type-short name>.h

RTE_<SWC-Type-short name>.h

Figure 12-8: E2E File dependencies

[UC_E2E_00239] The E2E Protection Wrapper, for the given SW-C identified with <SWC-Type-short name>, shall be made of two files: E2EPW_<SWC-Type-short name>.c and E2EPW_<SWC-Type-short name>.h.] (SRS_E2E_08528)

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

[UC_E2E_00241] E2EPW_<SWC-Type-short name>.h shall include used header files from E2E Library (used E2E_PXX.h files) and shall include RTE_<SWC-Type-short name>.h.] (SRS_E2E_08528)

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

[UC_E2E_00256] The E2E Protection Wrapper shall ensure the integrity of the safety-related data elements.] (SRS_E2E_08528)
[UC_E2E_00257] The implementation of the E2E Protection Wrapper (as a safety-related Software Component) shall comply with the requirements for the development of safety-related software for the automotive domain.
(SRS_E2E_08528)

12.1.5 Methodology

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

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

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

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

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

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

Most of the settings are defined under Software Component Template [11].
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 12-9: Release R4.0.1 and newer: E2E Protection Wrapper configuration (hardcopy from DOC_EndtoEndProtection)
The important settings are:

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

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

3. EndToEndProtectionISignalIPdu: association of one E2E protection to a one I-PDU and to one signal group,
   a. EndToEndProtectionISignalIPdu.dataOffset: offset in bits of the signal group in the I-PDU.
It is possible to add several signal groups into one I-PDU using several EndToEndProtectionISignalIPdu elements. The ISignalIPdu.unusedBitPattern is used by COM to create the final I-PDU and by E2E Protection Wrapper, to create a correct I-PDU representation of the protected data (on which a correct CRC can be computed).

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

![Diagram of DataPrototype](image)

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

The wrapper uses the standard E2E error codes of E2E library functions, which are extended with additional error codes.
[UC_E2E_0302]:
Where applicable, the following error status shall be used by E2E Wrapper functions within byte 3 of the return value, in addition to the error codes already defined by [SWS_E2E_00047] (chapter 7.2.1):

<table>
<thead>
<tr>
<th>Type or error or status</th>
<th>How should the caller of E2E Wrapper handle it</th>
<th>Related code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>Production</td>
<td>E2EPW_STATUS_OK</td>
<td>0x0</td>
</tr>
<tr>
<td>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.</td>
<td>Production</td>
<td>E2EPW_STATUS_NONEW_DATA</td>
<td>0x1</td>
</tr>
<tr>
<td>Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/Corrupted. This may be caused by corruption, insertion or by addressing faults.</td>
<td>Production</td>
<td>E2EPW_STATUS_WRONG_CRC</td>
<td>0x2</td>
</tr>
<tr>
<td>NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.</td>
<td>Production</td>
<td>E2EPW_STATUS_SYNC</td>
<td>0x3</td>
</tr>
<tr>
<td>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.</td>
<td>Production</td>
<td>E2EPW_STATUS_INITIALIZED</td>
<td>0x4</td>
</tr>
<tr>
<td>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.</td>
<td>Production</td>
<td>E2EPW_STATUS_REPEATED</td>
<td>0x8</td>
</tr>
<tr>
<td>OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 &lt; 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.</td>
<td>Production</td>
<td>E2EPW_STATUS_OKSOMELOST</td>
<td>0x20</td>
</tr>
<tr>
<td>Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter &gt; 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.</td>
<td>Production</td>
<td>E2EPW_STATUS_WRONG_SEQUENCE</td>
<td>0x40</td>
</tr>
</tbody>
</table>

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

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

[UC_E2E_0303]:
Where applicable, the following error flags shall be used by E2E Wrapper functions on byte 1 of the return value, in addition to the error codes already defined by [SWS_E2E_00047] (chapter 7.2.1):

<table>
<thead>
<tr>
<th>Type or error or status</th>
<th>How should the caller of E2E Wrapper handle it</th>
<th>Related code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>Integration or production</td>
<td>E2EPW_E_DESERIALIZATION</td>
<td>0x3</td>
</tr>
<tr>
<td>The control fields computed by Write1 and Write2 are not equal, i.e.</td>
<td>Integration or production</td>
<td>E2EPW_E_REDUNDANT</td>
<td>0x5</td>
</tr>
</tbody>
</table>
status of voting between Write1 and Write2 failed

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

[SWS_E2E_00314] The caller of the E2E Wrapper functions should handle the errors/statii defined in UC_E2E_0302 and UC_E2E_0303 according to the column “How do caller of E2E shall handle it”.J (SRS_E2E_08528)

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

12.1.7 E2E Protection Wrapper routines

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

[UC_E2E_00293] The parameter <instance> of the E2E Protection Wrapper routines shall be present if and only if the calling software component is multiply instantiated. Because in the current release multiple instantiation of software components is not supported by E2E Protection wrapper, this means that the optional parameter <instance> shall never be present.1 (SRS_E2E_08528)

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

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

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

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

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

12.1.7.1.1 E2EPW_Write_<p>_<o>}

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2EPW_Write_&lt;p&gt;_&lt;o&gt;</th>
</tr>
</thead>
</table>
| Syntax:     | `uint32 E2EPW_Write_<p>_<o>(
|             | Rte_Instance <instance>,
|             | -- <data> )` |
| Service ID[hex]: | 0 |
| 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: 

- 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_SEQ_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_INTEERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or postcondition)
  - E2E_E_OK - Function E2EPW_Write completed successfully

- The byte 2 is the return value of E2E_PXXProtect function:
  - E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer
  - E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range
  - E2E_E_INTEERR - An internal error has occurred in E2E_PXXProtect
The byte 3 is a placeholder for future use and takes the following values:

- **E2E_E_OK** - default case

**Description:**
Initiates a safe explicit sender-receiver transmission of a safety-related data element with data semantic. It protects data with E2E Library function E2E_PXXProtect and then it calls the corresponding RTE_Write function.

---

**Description:**

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

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

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

---

**[UC_E2E_00300]**

**Service name:** E2EPW_WriteInit_<p>_<o>

**Syntax:**

```
Std_ReturnType E2EPW_WriteInit_<p>_<o>(
    Rte_Instance <instance>
)
```

**Service ID[hex]:** 0x15

**Sync/Async:** Synchronous

**Reentrancy:** Non Reentrant

**Parameters (in):**

- `<instance>` SW-C instance. This parameter is not used (it is ignored).

**Parameters (inout):** None

**Parameters (out):** None

**Return value:**

- **Std_ReturnType**
  - **E2E_E_INTERR** - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition)
  - **E2E_E_OK** - Function completed successfully

**Description:**

The function reinitializes the corresponding data structure after a detected error or at startup.

---

**[UC_E2E_00301]**

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

- **Counter = 0**

(SRS_E2E_08528)
### 12.1.7.1.3 E2EPW_Read_<p>_<o>

**Service name:** E2EPW_Read_<p>_<o>

**Syntax:**
```c
uint32 E2EPW_Read_<p>_<o>(
    Rte_Instance <instance>,
    -- <data>
)
```

**Service ID[hex]:** 0

**Sync/Async:** Synchronous

**Reentrancy:** Non Reentrant

**Parameters (in):**
- `<instance>`: SW-C instance. This parameter is passed to the corresponding Rte_Read function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Read function.

**Parameters (inout):** None

**Parameters (out):**
- `<data>`: Parameter to pass back the received data. The pointer to the OUT parameter `<data>` must remain valid until the function call returns.

**Return value:**
- `uint32`: The byte 0 (lowest byte) is the status of Rte_Read function:
  - RTE_E_INVALID - data element invalid
  - RTE_E_MAX_AGE_EXCEEDED - data element outdated
  - RTE_E_NEVER_RECEIVED - No data received since system start or partition restart
  - RTE_E_UNCONNECTED - Indicates that the receiver port is not connected.
  - RTE_E_OK - data read successfully

  The byte 1 is the status of runtime checks done within E2E Protection Wrapper function, plus including bit extension checks:
  - E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer
  - E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range
  - E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or postcondition)
  - E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0.
  - E2E_E_OK - Function E2EPW_Read completed successfully

  The byte 2 is the return value of E2E_PXXCheck function:
  - E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXCheck is a NULL pointer
  - E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXCheck is erroneous, e.g. out of range
  - E2E_E_INTERR - An internal error has occurred in E2E_PXXCheck (e.g. error detected by program flow monitoring, violated invariant or postcondition)
  - E2E_E_OK - Function E2E_PXXCheck completed successfully

  The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function.
E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.

E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.

E2EPW_STATUS_INITIAL - Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.

E2EPW_STATUS_REPEATED - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL_ _OK_ or _OKSOMELOST.

E2EPW_STATUS_OK - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL_ _OK_ or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.

E2EPW_STATUS_OKSOMELOST - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL_ _OK_ or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.

E2EPW_STATUS_WRONGSEQUENCE - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL_ _OK_ or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.

E2EPW_STATUS_SYNC - NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.

Description:
Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.

SRS_E2E_08528

[UC_E2E_00192] The function E2EPW_Read_<p>_<o>() shall:
1. Invoke Rte_Read_<p>_<o>()
2. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
3. Invoke E2E Library function E2E_PXXCheck()
4. Do the deserialization check.

SRS_E2E_08528

See also Figure 12-14: E2EPW_Read sequence diagram and Figure 12-15: E2EPW_Read activity diagram.
12.1.7.1.4 E2EPW_ReadInit_<p>_<o>

**[UC_E2E_00296]**

<table>
<thead>
<tr>
<th>Service name</th>
<th>E2EPW_ReadInit_&lt;p&gt;_&lt;o&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Std_ReturnType E2EPW_ReadInit_&lt;p&gt;_&lt;o&gt;(</td>
</tr>
<tr>
<td></td>
<td>Rte_Instance &lt;instance&gt; )</td>
</tr>
<tr>
<td>Service ID[hex]</td>
<td>0x16</td>
</tr>
<tr>
<td>Sync/Async</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in)</td>
<td>&lt;instance&gt; SW-C instance. This parameter is not used (it is ignored).</td>
</tr>
<tr>
<td>Parameters (inout)</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out)</td>
<td>Std_ReturnType</td>
</tr>
<tr>
<td>Return value</td>
<td>Status of runtime checks:</td>
</tr>
<tr>
<td></td>
<td>E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition)</td>
</tr>
<tr>
<td></td>
<td>E2E_E_OK - Function completed successfully</td>
</tr>
<tr>
<td>Description</td>
<td>The function reinitializes the corresponding data structure after a detected error or at startup.</td>
</tr>
</tbody>
</table>

(SRS_E2E_08528)

The function E2EPW_ReadInit_<p>_<o> shall initialize the E2E_PXXCheckStateType_<p>_<o> with the following values:
- LastValidCounter = 0
- MaxDeltaCounter = 0
- WaitForFirstData = TRUE
- NewDataAvailable = FALSE
- LostData = 0
- Status = E2E_PXXSTATUS_NONEWDATA
- NoNewOrRepeatedDataCounter = 0
- SyncCounter = 0

12.1.7.2 Redundant wrapper routines

12.1.7.2.1 E2EPW_Write1_<p>_<o>

**[UC_E2E_00261]**

<table>
<thead>
<tr>
<th>Service name</th>
<th>E2EPW_Write1_&lt;p&gt;_&lt;o&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>uint32 E2EPW_Write1_&lt;p&gt;_&lt;o&gt;(</td>
</tr>
<tr>
<td></td>
<td>Rte_Instance &lt;instance&gt;,</td>
</tr>
<tr>
<td></td>
<td>-- &lt;data&gt; )</td>
</tr>
<tr>
<td>Service ID[hex]</td>
<td>0</td>
</tr>
<tr>
<td>Sync/Async</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in)</td>
<td>&lt;instance&gt; 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</td>
</tr>
</tbody>
</table>
### Parameters (inout):

<table>
<thead>
<tr>
<th>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.</th>
</tr>
</thead>
</table>

### Parameters (out):

| uint32 | The byte 0 (lowest byte) is equal to E2E_E_OK (because Rte_Write is not invoked) |
|---|
| None | The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: |
| | E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer |
| | E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range |
| | E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or postcondition) |
| | E2E_E_OK - Function E2EPW_Write completed successfully |
| | The byte 2 is the return value of E2E_PXXProtect function: |
| | E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer |
| | E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range |
| | E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition) |
| | E2E_E_OK - Function E2E_PXXProtect completed successfully |
| | The byte 3 is a placeholder for future use and takes the following values: |
| | E2E_E_OK - default case |

### Return value:

<table>
<thead>
<tr>
<th>It protects data with E2E Library function E2E_PXXProtect. it does not call the corresponding RTE_Write function.</th>
</tr>
</thead>
</table>

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

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

See also Figure 12-19: E2EPW_Write1 activity diagram.

### 12.1.7.2.2 E2EPW_Write2_<p>_<o>

[UC_E2E_00263]
### Specification of SW-C End-to-End Communication Protection Library

AUTOSAR CP Release 4.3.1

#### Service ID[hex]:
<instance>0

#### Sync/Async:
Sync/Async: Synchronous

#### Reentrancy:
Reentrancy: Non Reentrant

#### Parameters (in):
- **<data>**
  - 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.

- **<instance>**
  - Data element to be protected and sent. The parameter is IN, because this function compares the calculated protection fields from E2EPW_Write1 with independently calculated fields from invoking E2E_PXXProtect. Nothing is changed in <data> in case of success. The name and data type are the same as in the corresponding RTE_Write function.

#### Parameters (inout):
None

#### Parameters (out):
None

#### Return value:
- **uint32**
  - The byte 0 (lowest byte) is the status of RTE_Write function:
    - RTE_E_COM_STOPPED - the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only)
    - RTE_E_SEG_FAULT - a segmentation violation is detected in the handed over parameters to the RTE API. No transmission is executed
    - RTE_E_OK - data passed to communication service successfully

  The byte 1 is the status of runtime Protects done within E2E Protection Wrapper function:
    - E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer
    - E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range
    - E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or postcondition)
    - E2EPW_E_REDUNDANCY - The control fields computed by Write1 and Write2 are not equal, i.e. status of voting between Write1 and Write2 failed
    - E2E_E_OK - Function E2EPW_Write completed successfully

  The byte 2 is the return value of E2E_PXXProtect function:
    - E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXProtect is a NULL pointer
    - E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXProtect is erroneous, e.g. out of range
    - E2E_E_INTERR - An internal error has occurred in E2E_PXXProtect (e.g. error detected by program flow monitoring, violated invariant or postcondition)
    - E2E_E_OK - Function E2E_PXXProtect completed successfully

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

#### Description:
- Initiates a safe explicit sender-receiver transmission of a safety-related data element with data semantic. It protects data with E2E Library function E2E_PXXProtect, compares the computed control fields with the ones computed by Write1, and then it calls the corresponding RTE_Write function.
The function E2EPW_Write2_<p>_<o>() shall:
1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU
2. Invoke E2E Library function E2E_PXXProtect()
3. Execute voting on control fields between Write1 and Write2
4. Invoke Rte_Write_<p>_<o>()

See also Figure 12-20: E2EPW_Write2 activity diagram.

12.1.7.2.3 E2EPW_WriteInit1_<p>_<o>

Service name: E2EPW_WriteInit1_<p>_<o>
Syntax: uint8 E2EPW_WriteInit1_<p>_<o>(Rte_Instance <instance>)
Service ID[hex]: 0x17
Sync/Async: Synchronous
Reentrancy: Non Reentrant
Parameters (in): <instance> SW-C instance. This parameter is not used (it is ignored).
Parameters (inout): None
Parameters (out): None
Return value: uint8
  - 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.

The function E2EPW_WriteInit1_<p>_<o> shall initialize the E2E_PXXProtectStateType_<p>_<o> related to redundant channel 1 with the following values:
   Counter = 0.

12.1.7.2.4 E2EPW_WriteInit2_<p>_<o>

Service name: E2EPW_WriteInit2_<p>_<o>
Syntax: uint8 E2EPW_WriteInit2_<p>_<o>(Rte_Instance <instance>)
Service ID[hex]: 0x18
Sync/Async: Synchronous
Reentrancy: Non Reentrant
Parameters (in): None
Parameters (inout): None
Parameters (out): None
Return value: uint8
  - 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.
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.

[SWS_E2E_00323] The function E2EPW_WriteInit2_<p>_<o> shall initialize the E2E_PXXProtectStateType_<p>_<o> related to redundant channel 2 with the following values:
- `Counter = 0` (SRS_E2E_08528)

12.1.7.2.5 E2EPW_Read1_<p>_<o>

[UC_E2E_00265] [UC_E2E_00265]

Service name: E2EPW_Read1_<p>_<o>

Syntax: `uint32 E2EPW_Read1_<p>_<o>(
     Rte_Instance `<instance>',
     -- `<data>
)

Service ID[hex]: 0
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.NeverReceived` - No data received since system start or partition restart
- `RTE_E_UNCONNECTED` - Indicates that the receiver port is not connected.
- `RTE_E_OK` - data read successfully

The byte 1 is the status of runtime checks done within E2E Protection Wrapper function:
- `E2E_E_INPUTERR_NULL` - At least one pointer parameter of E2EPW_Read is a NULL pointer
- `E2E_E_INPUTERR_WRONG` - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range
E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or postcondition)

E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0.

E2E_E_OK - Function E2EPW_Read completed successfully

The byte 2 is the return value of E2E_PXXCheck function:
E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXCheck is a NULL pointer
E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXCheck is erroneous, e.g. out of range
E2E_E_INTERR - An internal error has occurred in E2E_PXXCheck (e.g. error detected by program flow monitoring, violated invariant or postcondition)
E2E_E_OK - Function E2E_PXXCheck completed successfully

The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function.
E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.
E2EPW_STATUS_INITIAL - Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver’s initialization or reinitialization, so the Counter cannot be verified yet.
E2EPW_STATUS_REPEATED - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
E2EPW_STATUS_OK - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
E2EPW_STATUS_OKSOMELOST - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
E2EPW_STATUS_WRONGSEQUENCE - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
E2EPW_STATUS_SYNC - NOT VALID: The new data has been
**Specification of SW-C End-to-End Communication Protection Library**

AUTOSAR CP Release 4.3.1

---

<table>
<thead>
<tr>
<th>Specification of SW-C End-to-End Communication Protection Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR CP Release 4.3.1</td>
</tr>
</tbody>
</table>

- AUTOSAR confidential -

Document ID 428: AUTOSAR_SWS_E2ELibrary

---

**Description:**

Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.

\[ (SRS\_E2E\_08528) \]

**[UC_E2E_00266]**

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

1. Invoke Rte_Read_<p>_<o>()
2. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
3. Invoke E2E Library function E2E_PXXCheck()
4. Do the deserialization check.\[ (SRS\_E2E\_08528) \]

See also Figure 12-16: E2EPW_Read1 activity diagram.

---

**12.1.7.2.6 E2EPW_Read2_<p>_<o>**

**[UC_E2E_00267] [**

<table>
<thead>
<tr>
<th>Service name:</th>
<th>E2EPW_Read2_&lt;p&gt;_&lt;o&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>uint32 E2EPW_Read2_&lt;p&gt;_&lt;o&gt;(</td>
</tr>
<tr>
<td></td>
<td>Rte_Instance &lt;instance&gt;,</td>
</tr>
<tr>
<td></td>
<td>-- &lt;data&gt; )</td>
</tr>
<tr>
<td><strong>Service ID[hex]:</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Sync/Async:</strong></td>
<td>Synchronous</td>
</tr>
<tr>
<td><strong>Reentrancy:</strong></td>
<td>Non Reentrant</td>
</tr>
</tbody>
</table>

**Parameters (in):**

- **<instance>**
  - SW-C instance. This parameter is passed to the corresponding Rte_Read function, and apart from that the parameter is unused by E2E Protection Wrapper. This means that the wrapper ignores the instance of SW-C. The name and data type are the same as in the corresponding Rte_Read function.

- **<data>**
  - The received data to be checked. The parameter is IN, because this function re-performs the checks on the already received data (by E2EPW_Read1_<p>_<o>). Nothing is changed in <data>. The pointer to the IN parameter <data> must remain valid until the function call returns.

**Parameters (inout):**

None

**Parameters (out):**

None

**Return value:**

- **uint32**
  - The byte 0 (lowest byte) equal to RTE_E_OK (because Rte_Read is not invoked)
  - The byte 1 is the status of runtime checks done within E2E Protection Wrapper function:
    - E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer
    - E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range
    - E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or
postcondition)
E2EPW_E_DESERIALIZATION - extension/expansion error(s) occurred. It is the status if bit extension (conversion of shortened I-PDU representation into data elements) is correct. For example, if 12 bits from I-PDU are expanded into 16-bit uint, then the top most 4 bits shall be 0.
E2E_E_OK - Function E2EPW_Read completed successfully

The byte 2 is the return value of E2E_PXXCheck function:
E2E_E_INPUTERR_NULL - At least one pointer parameter of E2E_PXXCheck is a NULL pointer
E2E_E_INPUTERR_WRONG - At least one input parameter of E2E_PXXCheck is erroneous, e.g. out of range
E2E_E_INTERR - An internal error has occurred in E2E_PXXCheck (e.g. error detected by program flow monitoring, violated invariant or postcondition)
E2E_E_OK - Function E2E_PXXCheck completed successfully

The byte 3 is the value of E2E_PXXCheckStatusType Enumeration, representing the result of the verification of the Data in E2E Profile XX, determined by the Check function.
E2EPW_STATUS_NONEWDATA - Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
E2EPW_STATUS_WRONGCRC - Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.
E2EPW_STATUS_INITIALIZER - Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.
E2EPW_STATUS_REPEATED - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
E2EPW_STATUS_OK - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
E2EPW_STATUS_OKSOMELOST - OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
E2EPW_STATUS_WRONGSEQUENCE - Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
E2EPW_STATUS_SYNC - NOT VALID: The new data has been received after detection of an unexpected behaviour of counter. The data has a correct CRC and a counter within the expected range with
The function re-checks the data received with corresponding function Read1 by means of execution of E2E_PXXCheck.

[SRS_E2E_08528]

[UC_E2E_00268] The function E2EPW_Read2_<p>_<o>() shall:
1. If this communication is inter-ECU and the Data element is not an opaque uint8 byte array, then serialize the data element into the layout identical to the one of the corresponding area in I-PDU.
2. Invoke E2E Library function E2E_PXXCheck()
3. Do the deserialization check.

See also Figure 12-17: E2EPW_Read2 activity diagram.

12.1.7.2.7 E2EPW_ReadInit1_<p>_<o>

[SWS_E2E_00320] [SWS_E2E_00324]

Service name: E2EPW_ReadInit1_<p>_<o>

Syntax:
uint8 E2EPW_ReadInit1_<p>_<o>(
    Rte_Instance <instance>
)

Service ID[hex]: 0x19
Sync/Async: Synchronous
Reentrancy: Non Reentrant
Parameters (in): <instance> - SW-C instance. This parameter is not used (it is ignored).
Parameters (inout): None
Parameters (out): None

Return value:
uint8 The byte 0 is the status of runtime checks:
E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or postcondition)
E2E_E_OK - Function completed successfully

Description: The function reinitializes the corresponding data structure after a detected error or at startup.

[SRS_E2E_08528]

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

LastValidCounter = 0
MaxDeltaCounter = 0
WaitForFirstData = TRUE
NewDataAvailable = FALSE
LostData = 0
Status = E2E_PXXSTATUS_NONEWDATA
NoNewOrRepeatedDataCounter = 0
SyncCounter = 0.

[SRS_E2E_08528]
12.1.7.2.8 E2EPW_ReadInit2_<p>_<o>

[SWS_E2E_00321]

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

[SWS_E2E_00325] The function E2EPW_ReadInit2_<p>_<o> shall initialize the  
E2E_PXXCheckStateType_<p>_<o> related to redundant channel 2 with the  
following values:  
LastValidCounter = 0  
MaxDeltaCounter = 0  
WaitForFirstData = TRUE  
NewDataAvailable = FALSE  
LostData = 0  
Status = E2E_PXXSTATUS_NONEWDATA  
NoNewOrRepeatedDataCounter = 0  
SyncCounter = 0. | (SRS_E2E_08528)
12.1.8 E2EPW Routines Diagrams

12.1.8.1 Sequence Diagrams – Read and Write
Step S0: Application writes some values in each primitive element speed and accel of the complex data element AppDataEl. AppDataEl also contains primitive elements crc for CRC and counter for Counter, used for End-to-End Communication Protection.

Step S1: SWC invokes E2EPWWrite_<p>_<o>(..) instead of RTE_Write_<p>_<o>(..), using the same parameters.

Step S3: SWC interface wrapper calls E2E_PXXProtect function, passing the data element configuration, state and serialized representation of the data element.

Step S4: E2E library executes protection, updates State and AppDataEl.

Step S5: Write protection information back to data element (CRC, Counter; DataIdNibble, if applicable).

Step S6: E2E library calls RTE function to send the data element.

Step S7: E2E library creates the return value and returns.

Step S8: Return value is returned to SWC. The value is composed of the return values of E2E_PXXProtect(..) and RTE_Write(..).

Figure 12-13: E2EPW_Write sequence diagram
Step R1: SWC invokes E2EPW_Read_<p>_<o>(..) instead of RTE_Read_<p>_<o>(..), using the same parameters.

Step R2: E2EPW invokes RTE_Read interface, getting the most current received data element.

Step R3: NewDataAvailable is set to TRUE if Rte_Read returned with no error.

Step R4: Serialize data element to I-PDU representation.

Step R5: Call E2E_PXXCheck().

Step R6: E2E library computes CRC, executes checks.

Step R7: Perform Deserialization check.

Step R8: Combine return and status values.

Step R9: Decide if DataEl can be used or if a fault reaction must be triggered.

Check return value

Verification that deserialization by COM was correctly done (sign extension/value ranges):

Ret = combination of (return) values Rte_Read_<p>_<o>(..), E2E_PXXCheck(..), State->Status, deserialization valid

Check CRC, Counter, NewDataAvailable update State

Step R10: State->NewDataAvailable = (retRteRead == RTE_E_OK) ? TRUE : FALSE

Invoke CRC Library multiple times to calculate CRC of Data and DataID: crc

Check CRC, Counter, NewDataAvailable update State

Verify that deserialization by COM was correctly done (sign extension/value ranges): bool

Ret = combination of (return) values Rte_Read_<p>_<o>(..), E2E_PXXCheck(..), State->Status, deserialization valid

E2EPW_Read_<p>_<o>(&dataEl)

Rte_Read_<port>_<data>(&DataEl)

retRteRead

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

Serialize data element to I-PDU representation

E2E_PXXCheck(&Config, &State, &Data)

Invoke CRC Library multiple times to calculate CRC of Data and DataID: crc

Check CRC, Counter, NewDataAvailable update State

Verify that deserialization by COM was correctly done (sign extension/value ranges): bool

Ret = combination of (return) values Rte_Read_<p>_<o>(..), E2E_PXXCheck(..), State->Status, deserialization valid

retE2ECheck

Check return value

Figure 12-14: E2EPW_Read sequence diagram
12.1.8.2 Activity Diagrams – E2EPW Read, Read1 and Read2

E2EPW_Read_<p>_<o> (AppDataEl)

WriteVariable
retRteRead = RTE_E_OK;
plausibilityChecks = E2E_E_OK;
retE2ECheck = E2E_E_OK;

AppDataEl == NULL
[TRUE] [FALSE]

WriteVariable
rteRead = Rte_Read_<p>_<o> (AppDataEl)
(\)
retRteRead == RTE_E_OK
[TRUE] [FALSE]

WriteVariable
State->NewDataAvailable = TRUE
WriteVariable
State->NewDataAvailable = FALSE

Check deserialization of AppDataEl
Deserialization invalid?
[TRUE] [FALSE]

WriteVariable
plausibilityChecks = E2EPW_E_DESERIALIZATION

WriteVariable
ret = ( retRteRead | (retE2ECheck<<8) | (plausibilityChecks<<16) | (State->Status<<24) )

return ret

Figure 12-15: E2EPW_Read activity diagram
Figure 12-16: E2EPW_Read1 activity diagram
Figure 12-17: E2EPW_Read2 activity diagram
12.1.8.3 Activity Diagrams – E2EPW Write, Write1 and Write2
E2EPW_Write_<p>_<o> (AppDataEl)

WriteVariable
retRteWrite = RTE_E_OK;
retE2EProtect = E2E_E_OK;
plausibilityChecks = E2E_E_OK;

AppDataEl == NULL

[TRUE]

WriteVariable
plausibilityChecks = E2E_E_INPUTERR_NULL

[FALSE]

SerializeAppDataEl to IPDU representation
Data if AppDataEl is complex data element;
else it is a byte array, copy it to Data.

retE2EProtect = E2E_PXXProtect
(Config, State, Data)
(\::\)

retE2EProtect == E2E_E_OK

[FALSE]

Write back CRC and
Counter from Data to
AppDataEl

retRteWrite = Rte_Write_<p>_<o>
(AppDataEl)
(\::\)

[TRUE]

WriteVariable
ret = retRteWrite |
(retE2EProtect<<8) |
(plausibilityChecks<<16)

return ret
Figure 12-18: E2EPW_Write activity diagram
E2EPW_Write1_<p>_<o> (AppDataEl)

WriteVariable
retRteWrite = RTE_E_OK;
retE2EProtect = E2E_E_OK;
plausibilityChecks = E2E_E_OK;

AppDataEl == NULL

[TRUE]

WriteVariable
plausibilityChecks = E2E_E_INPUTERR_NULL

[FALSE]

SerializeAppDataEl to IPDU representation
Data if AppDataEl is complex data element;
else it is a byte array, copy it to Data.

retE2EProtect = E2E_PXXProtect
(Config1, State1, Data)

[FALSE]

retE2EProtect == E2E_E_OK

[TRUE]

Write back CRC and
Counter from Data to
AppDataEl

WriteVariable
ret = retRteWrite |
(retE2EProtect<<8) |
(plausibilityChecks<<16)

return ret

Figure 12-19: E2EPW_Write1 activity diagram
E2EPW_Write2_<p>_<o> (AppDataEl)

WriteVariable
retRteWrite = RTE_E_OK;
retE2EProtect = E2E_E_OK;
plausibilityChecks = E2E_E_OK;

[TRUE] AppDataEl == NULL

[FALSE] SerializeAppDataEl to IPDU representation

Data if AppDataEl is complex data element; else it is a byte array, copy it to Data.

WriteVariable
plausibilityChecks = E2E_E_INPUTERR_NULL

retE2EProtect = E2E_PXXProtect (Config2, State2, Data) (::)

[FALSE] retE2EProtect == E2E_E_OK

[TRUE] Crc and Counter value in Data equal CRC and Counter value in data element

plausibilityChecks = E2E_PW_E_REDUNDANCY

[FALSE] retRteWrite = Rte_Write_<p>_<o> (AppDataEl) (::)

[TRUE]

WriteVariable
ret = retRteWrite | (retE2EProtect<<8) | (plausibilityChecks<<16)

return ret

Figure 12-20: E2EPW_Write2 activity diagram
12.1.9 Code Example

**Important:**

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

The suffixes used shall be:

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

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

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

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

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

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

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

1. Code redundancy:
   a. For each `Rte_Write*` function, there are corresponding `E2EPW_Write1*` and `E2EPW_Write2*` functions
   b. For each `Rte_Read*` function, there are corresponding `E2EPW_Read1*` and `E2EPW_Read2*` functions

2. Time diversity:
   a. The functions `E2EPW_Write1*` and `E2EPW_Write2*` on the sender side and `E2EPW_Read1*` and `E2EPW_Write2*` are executed one after each other.

3. Data redundancy:
   a. All data used by the redundant wrapper, apart from application data element, is redundant
   b. The application data element is instantiated by `Rte` one time only. To mitigate faults, is written/read by application at each call of `E2EPW_Write1`, `E2EPW_Write2`, `E2EPW_Read1`, `E2EPW_Read2`. 
There are no configuration options in AUTOSAR templates to select which wrapper shall be generated. Either redundant or single channel functions should be generated (generating both single channel and redundant wrapper calls for the same SW-Cs would signify generation of dead code). The choice which wrapper is generated may be a global option in the wrapper generator. Alternatively, a wrapper may be able to generated either single-channel or redundant wrapper only.

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

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

12.1.9.1 Code Example – Sender SW-C

12.1.9.1.1 Sender – E2EPW_WriteInit, E2EPW_WriteInit1 and E2EPW_WriteInit2

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

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

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

/* byte array for call of E2Elib */
static uint8 Data_<p>_<o>[<DataLength / 8>];
```
Std_ReturnType E2EPW_WriteInit_<p>_<o>(Rte_Instance Instance) {
    State_<p>_<o>.Counter = 0;
    return E2E_E_OK;
}

For redundant wrapper:

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

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

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

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

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

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

Std_ReturnType E2EPW_WriteInit2_<p>_<o>(Rte_Instance Instance) {
    State2_<p>_<o>.Counter = 0;
    return E2E_E_OK;
}
12.1.9.1.2 Sender – E2EPW_Write and E2EPW_Write1

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

12.1.9.1.2.1 Generation / Initialization

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

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

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

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

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

12.1.9.1.2.2 Step S0

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

Case A (complex data type)

```c
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):

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

12.1.9.1.2.3 Step S1
Step S1: Application calls E2E Protection Wrapper.

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

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

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

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

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

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

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

/* Data has the same layout as serialized signal group in I-PDU.
   Initialize all bytes of Data[] with the unused bit pattern
   (called unusedBitPattern in system template. */
Data_<p>_<o>[0] = 0;
   /* in accel, only 4 bits are used, they go
     To high nibble of Data[1], next to Counter. */
   Data_<p>_<o>[1] = (AppDataEl->accel & 0x0F) << 4;
   /* in speed, only 8+4 bits are used.
     low byte of speed goes to Data[2]. */
   Data_<p>_<o>[2] = (AppDataEl->speed & 0x00FF);
   /* low nibble of high byte goes to Data[3] */
Data_<p>_<o>[3] = (AppDataEl-&gt;speed & 0x0F00) &gt;&gt; 8;
/* high nibble of high byte of Data[3] is unused, so it is set with 1s on each unused bit */
Data_<p>_<o>[3] |= 0xF0;
/* Data[4] is unused but transmitted, so it is explicitly set to 0xFF*/
Data_<p>_<o>[4] = 0xFF;

The above example is illustrated by the figure below:

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

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

Case B (array):
The E2E Protection Wrapper (E2EPW_Write_<p>_<o>, E2EPW_Writel_<p>_<o>()) simply casts the data element to the array and copies it:

Std_ReturnType plausibilityChecks = E2E_E_OK;
...
/* example of possible plausibility checks */
if (AppDataEl == NULL) {
    return (E2E_E_INPUTERR_NULL &lt; 8);
}
memcpy(Data_<p>_<o>, AppDataEl, 8);

12.1.9.1.2.5 Step S3
Step S3: E2E Protection Wrapper (E2EPW_Write_<p>_<o>, E2EPW_Writel_<p>_<o>()) calls the E2E library to protect the data element.

/* single channel - Write */
Std_ReturnType retE2EPprotect = E2E_P01Protect(&Config_<p>_<o>, &State_<p>_<o>, Data_<p>_<o>);

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

/* redundant - Writel */
Std_ReturnType retE2EPprotect = E2E_P01Protect(&Config1_<p>_<o>, &State1_<p>_<o>, Data_<p>_<o>);
12.1.9.1.2.6  Step S4

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

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

**Figure 12-22: Step 4**

12.1.9.1.2.7  Step S5

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

Case A (complex data type):

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

This is illustrated by the Figure 12-23:

```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*/
} AppDataEl;
```

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

Case B (array):

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

12.1.9.1.2.8  Step S6
Step S6: Single channel Wrapper \((E2EPW\_Write\_<p>\_<o>)\) calls RTE function to send the data element and returns the extended status to SW-C.

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

Redundant wrapper \((E2EPW\_Write\_<p>\_<o>)\) in step S7 does not call Rte_Write_<p>_<o>() function.

```c
/* Redundant - Write1 */
Std_ReturnType retRteWrite = E2E\_E\_OK;
```

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

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

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

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

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

12.1.9.1.3 Sender - E2EPW_Write2

This chapter presents an example implementation of function \(E2EPW\_Write2\_<p>\_<o>()\).

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

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

12.1.9.1.3.2 Step S11

Steps S11-S18 represent the steps of the function \(E2EPW\_Write2\_<p>\_<o>()\).
Step S11: Application calls E2E Protection Wrapper for the second time, this time E2EPW_Write2_<p>_<o>() function.

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

12.1.9.1.3.3 Step S12
The step S13 (of function E2EPW_Write2_<p>_<o>()) is 100% identical to Step S2 (of function E2EPW_Write1_<p>_<o()>).

12.1.9.1.3.4 Step S13
Step S3: E2E Protection Wrapper (E2EPW_Write2_<p>_<o>()) calls the E2E library to protect the data element.

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

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

12.1.9.1.3.6 Step S15 – skipped
Contrary to Step S5, there is no copying back of control fields back to AppDataEl in E2EPW_Write2_<p>_<o>().

12.1.9.1.3.7 Steps S16
At this stage, the Wrapper (E2EPW_Write2_<p>_<o>()) has to its disposition the following:

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

There are two safety mechanisms provided:

1. The control fields (crc and counter from **AppDataEl** and from **Data**) are binary compared by the voter. By this means, the results Write1 and Write2 are voted by the sender.
2. 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):

```c
if( (AppDataEl->counter != (Data_port1_del[1] & 0x0F)) ||
    (AppDataEl->crc != (Data_port1_del[0] & 0x0F)))
plausibilityChecks = E2EPW_E_REDUNDANCY; /* 0x05 */

Std_ReturnType retRteWrite = E2E_E_OK;
/* Write data regardless if redundancy error detected ... */
retRteWrite = Rte_Write_<p>_<o>(Instance, AppDataEl);
```

Case B (array):

```c
if( ((AppDataEl[1] & 0x0F) != (Data_port1_del[1] & 0x0F)) ||
    (AppDataEl[0] != (Data_port1_del[0] & 0x0F)))
plausibilityChecks = E2EPW_E_REDUNDANCY; /* 0x05 */

Std_ReturnType retRteWrite = E2E_E_OK;
/* Write data regardless if redundancy error detected ... */
retRteWrite = Rte_Write_<p>_<o>(Instance, AppDataEl);
```

12.1.9.1.3.8  Step S17

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

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

12.1.9.1.3.9  Step S18

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

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

12.1.9.2  Code Example – Receiver SW-C

12.1.9.2.1 Receiver - E2EPW_ReadInit, E2EPW_ReadInit1 and E2EPW_ReadInit2

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

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

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

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

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

For redundant wrapper:

```c
static const E2E_P01ConfigType Config1_<p>_<o> =
{ 8, /* CounterOffset */
  0, /* CRCOffset */
  0x12, /* DataID */
  12, /* DataIDNibbleOffset */
  E2E_P01_DATAID_BOTH, /* DataIDMode */
  64, /* DataLength */
  1, /* MaxDeltaCounterInit */
  3, /* MaxNoNewOrRepeatedData */
  2, /* SyncCounterInit */
};
```
static const E2E_P01ConfigType Config2_{<p>_<o> =
    { 8,  /* CounterOffset */
      0,  /* CRCOffset */
      0x12,  /* DataID */
      12,  /* DataIDNibbleOffset */
      E2E_P01_DATAID_BOTH,  /* DataIDMode */
      64,  /* DataLength */
      1,  /* MaxDeltaCounterInit */
      3,  /* MaxNoNewOrRepeatedData */
      2,  /* SyncCounterInit */
    }
;

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

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

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

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

Std_ReturnType E2EPW_ReadInit2_{<p>_<o>(Rte_Instance Instance) {
    State2_{<p>_<o>.LastValidCounter = 0;
    State2_{<p>_<o>.MaxDeltaCounter = 0;
    State2_{<p>_<o>.WaitForFirstData = TRUE;
    State2_{<p>_<o>.NewDataAvailable = FALSE;
12.1.9.2.2 Receiver - E2EPW_Read and E2EPW_Read1

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

12.1.9.2.2.1 Generation / Initialization

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

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

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

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

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

12.1.9.2.2.2 Step R1

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

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

/* redundant – Read1 */
uint32 wrapperRet1 = E2EPW_Read1_<p>_<o>(Instance, AppDataEl);
```
12.1.9.2.2.3 Step R2
Step R2: Wrapper (E2EPW_Read_<p>_<o>, E2EPW_Read1_<p>_<o>()) checks the parameters and then calls RTE function Rte_Read to receive the data element.

```c
Std_ReturnType plausibilityChecks = E2E_E_OK, retRteRead;
...
/* example of possible plausibility checks */
if (AppDataEl == NULL) {
    return (E2E_E_INPUTERR_NULL);
}
retRteRead = Rte_Read_<p>_<o>(Instance, AppDataEl);
```

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

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

Redundant wrapper:

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

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

Case A (complex data type):

```c
/* For storing the same layout as the one of I-PDU */
Data_<p>_<o>[0] = 0;
/* in accel, only 4 bits are used,
    they go To high nibble of Data[1], next to Counter. */
Data_<p>_<o>[1] = (AppDataEl->accel &0x0F) << 4;
/* in speed, only 8+4 bits are used.
    low byte of speed goes to Data[2]. */
Data_<p>_<o>[2] = (AppDataEl->speed & 0x00FF);
/* low nibble of high byte goes to Data[3] */
Data_<p>_<o>[3] = (AppDataEl->speed & 0xF00) >> 8;
```
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:

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

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

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

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

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

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

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

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

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

Case A (complex data type):
The E2E Protection Wrapper verifies that the bit extensions done by COM are done correctly. This step is needed, because unused most significant bits of primitive data elements are simply cut out (not placed in I-PDUs). On the receiver side, these unused bits shall have a specified value (e.g. they shall be 0 for unsigned numbers). Note that the unused most significant bits of signals are not related to unused bits between signals in I-PDUs.
/* in accel, only 4 bits are used, they go
   To high nibble of Data[1], next to Counter. */

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

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

Case B (array):
Not present, as there is no bit extension done by COM

plausibilityChecks = E2E_E_OK;

12.1.9.2.2.8 Step R8
Step R8: The E2E wrapper returns to the application.

/* single channel */
return ( (retRteRead) | (retE2ECheck<<16) |
       (plausibilityChecks<<8) | (uint32)(State_p_o.Status)<<24 );

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

/* redundant */
return ( (retRteRead) | (retE2ECheck<<16) |
         (plausibilityChecks<<8) | (uint32)(State1_p_o.Status)<<24 );

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

Case A (complex data type):
/* single channel */
if( ((wrapperRet )&0xFF != 0) ||
    ((wrapperRet>>8 )&0xFF != 0) ||
    ((wrapperRet>>16)&0xFF != 0) ||
    (((wrapperRet>>24)&0xFF != E2EPW_STATUS_OKSOMELOST) &&
    ((wrapperRet>>24)&0xFF != E2EPW_STATUS_OK))
)
    {
        swc_error_handler(wrapperRet);
    }
targetSpeed = AppDataEl->speed;
targetAccel = AppDataEl->accel;

/* redundant */
if( ((wrapperRet1) & 0xFF != 0) ||
    ((wrapperRet1>>8) & 0xFF != 0) ||
    ((wrapperRet1>>16) & 0xFF != 0) ||
    ((wrapperRet1>>24) & 0xFF != E2EPW_STATUS_OKSOMELOST) &&
    ((wrapperRet1>>24) & 0xFF != E2EPW_STATUS_OK))
) {
    swc_error_handler(wrapperRet1);
}
targetSpeed1 = AppDataEl->speed;
targetAccel1 = AppDataEl->accel;

Case B (array):

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

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

/* redundant */
if( ((wrapperRet1) & 0xFF != 0) ||
    ((wrapperRet1>>8) & 0xFF != 0) ||
    ((wrapperRet1>>16) & 0xFF != 0) ||
    ((wrapperRet1>>24) & 0xFF != E2EPW_STATUS_OKSOMELOST) &&
    ((wrapperRet1>>24) & 0xFF != E2EPW_STATUS_OK))
) {
    swc_error_handler(wrapperRet1);
}
if(wrapperRet1 != 0) swc_error_handler(wrapperRet1);
uint16 targetSpeed1 = (AppDataEl[2]) | (AppDataEl[3]<<8 & 0x0F);
uint8 targetAccel1 = AppDataEl[1] >> 4;

12.1.9.2.3 Receiver - E2EPW_Read2

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

12.1.9.2.3.1  Step R10 – skipped
Value unused to numbering consistency.

### 12.1.9.2.3.2 Step R11
Step R11: Application calls the wrapper again.

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

### 12.1.9.2.3.3 Step R12 – partially skipped
Contrary to step R2 RTE is not read. Both read steps use the same data from RTE. There is only checking for parameters:

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

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

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

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

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

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

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

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

### 12.1.9.2.3.9 Step R18
Step R8: The E2E wrapper returns to the application.
return ( (retRteRead) | (retE2ECheck<<16) |
        (plausibilityChecks<<8) | (uint32)(State2.<p>.<o>.Status)<<24 );

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

Case A (complex data type):

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

/* check if E2EFW_Read2 was successful */
if(wrapperRet2 != 0) swc_error_handler(wrapperRet2);

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

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

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

Case B (array):

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

/* check if E2EFW_Read2 was successful */
if(wrapperRet2 != 0) swc_error_handler(wrapperRet2);

/* Check if both Read1 and Read2 report the same status. In particular, byte2 of ret1 and ret2 shall be identical. If not, then means that there is a disagreement on evaluation of data between Read1 and Read2 */
if(wrapperRet2 != wrapperRet1) swc_error_handlerR(wrapperRet1, wrapperRet2);
12.2 COM E2E Callouts

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

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

The callout invokes the E2E Library, once for each E2E-protected signal group in a given I-PDU.

This solution can be used in the systems where the integrity of operation of COM and RTE is provided.

12.2.1 Functional overview

For each I-PDU, there is a separate callout function. Each I-PDU callout function “knows” if and how each signal group of the I-PDU needs to be protected/checkered. This means that the callout invokes the E2E Library functions with appropriate settings and state parameters. The E2E Library does now “know” signal groups and their settings – entire information is passed as function parameters to E2E library functions.

On both receiver and sender side, if a callout returns TRUE, then COM continues. If a COM E2E Callout returns FALSE, then COM stops to process the given I-PDU (in this cycle). The COM E2E Callout returns FALSE if and only if there is an internal error, e.g. program flow error, data corruption error in E2E Lib.

The sender callout always TRUE if there are no runtime errors detected (e.g. wrong parameter), otherwise FALSE. The receiver callout receiver returns TRUE if there are no runtime errors detected and the result of the check is either E2E_P02STATUS_OK or E2E_P02STATUS_OKSOMELOST.

The diagram below summarizes the COM E2E Callout solution on the sender side. The SW-C is completely not impacted, and only additional activities in COM is invocation of the generated callout (step 6). If the return value from the callout is TRUE, then the IpduData modified by E2E Library is then transmitted by PDU router. If false, then COM stops further processing of this I-PDU in this cycle.
The diagram below summarizes the COM E2E Callout solution. The very important step is that the E2E Library overwrites CRC byte in the signal group by the check status bits (E2E_PXXCheckStateType). Then, this overwritten CRC byte is converted by COM to signals and then by RTE to data elements. As a result, the SW-C receives in the CRC data element the E2E check bits, and not the CRC value.

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

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

Sending/Calling
On the sender COM side, when the I-PDU has been built from signals and the conversions (e.g. Endianness) have taken place, and the I-PDU is ready, then COM calls a callout function. There is a separate callout for each I-PDU (if defined). Once the callout returns, COM invokes the PDU Router to transmit the data (function PduR_ComTransmit).

The callout function is generated to protect the signal groups of one I-PDU and simply invokes the E2E Library (once per each E2E-protected signal group) with the correct hard-coded settings. The hard-coded settings have been generated from the settings described in the previous section. When the callout returns TRUE, COM invokes PduR_ComTransmit(), to route the I-PDU through the network.

Figure 12-26: Callout – sequence – sending

According to COM SWS, the callouts shall conform to the following syntax:

```c
boolean <IPDU_CalloutName> (PduIdType TxPduId, const PduInfoType* PduInfoPtr)
```

**[UC_E2E_00250]** The transmission callout for usage with E2E shall be the following:

```c
IPDU_E2EPutected_<IPDU ID>(PduIdType TxPduId, PduInfoType* PduInfoPtr).
```
For example, the callout to protect the I-PDU with handle 21 shall have the name `IPDU_E2EProtect_21()` \(\text{(SRS_E2E_08528)}\)

**Reception**

On the receiver COM side, when the I-PDU is available at PDU Router, PDU Router invokes COM’s function `COM_RxIndication()`. COM then calls the generated I-PDU callout (if configured for the given I-PDU). The callout, generated specifically for that I-PDU, calls the E2E Library with specific parameters (once for each E2E-protected signal group). The E2E Library executes the checks and stores the check results in the status.

Once E2E Library check function returns, the callout copies the status into the CRC byte, so that it can be analyzed, if needed, by receiver SW-C.

![Figure 12-27: Callout - sequence - reception](image)

**[UC_E2E_00251]** The reception callout for usage with E2E shall be the following:

`IPDU_E2ECheck_<IPDU ID>(PduIdType RxPduId, PduInfoType* PduInfoPtr)`.

For example, the callout to protect the signal groups in an I-PDU with handle 21 shall have the name `IPDU_E2ECheck_21`() \(\text{(SRS_E2E_08528)}\)
12.2.2 Methodology

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

The information how each signal group needs to be protected (e.g. which E2E Profile, which offset) is defined in System Template [12], Software Component Template [11] and ECU configuration [13]. This configuration information is used to generate the callout functions.

By means of the settings defined by AUTOSAR templates, it is possible to generate the COM callouts for invoking the E2E Library.

The configuration is done in the following configuration areas:
1. Definition of I-PDUs (system template)
2. Definition of E2E settings (software component template)
3. Association of I-PDUs to E2E protection settings (system template).
4. Definition of I-PDU details (ECU configuration)

The four above steps are described in more details below.

First, according to System Template, the I-PDUs exchanged by COM are defined.

Secondly, according to Software Component Template, for each signal group to be protected, the classes EndToEndProtection and EndToEndDescription are defined. The settings include information like CRC offset.

Thirdly, according to System Template, each I-PDU to be protected is associated to a corresponding EndToEndProtection.

Fourth, after the extraction of ECU configuration, according to ECU configuration, the I-PDU handles (numerical I-PDU identifiers) and callout functions are defined. COM requires that there is a separate callout function for each I-PDU (separate piece of code).

All configuration options needed to generate the COM callouts automatically is available in AUTOSAR methodology. For each I-PDU to be protected/checked, a separate callout routine shall be generated, which invokes E2E Library (once or several times).

[UC_E2E_00270] The COM E2E callout shall be generated for the I-PDU for which the corresponding EndToEnd* metaclasses are defined. (SRS_E2E_08528)

[UC_E2E_00290] If the E2EProtection is done via COM Callouts then the EndToEndProtectionISignalIPdu shall be defined. (SRS_E2E_08528)
Note that in R3.2 (contrary to >=R4.0), the ISignalIPdu is called “SignalIPdu” and it inherits the unusedBitPattern attribute from IPdu.

The important settings are:

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

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

3. EndToEndProtectionISignalIPdu: association of one E2E protection to a one I-PDU and to one signal group,
a. EndToEndProtection

ISignalIPdu.dataOffset: offset in bits of the signal group in the I-PDU.

ISignalIPdu.unusedBitPattern is not used by E2E COM callouts, because they are set by COM and E2E COM callouts operate on the same buffers.

### 12.2.3 Code Example

Note that the code examples for the COM E2E callouts are for the case when there is one signal group in the I-PDU. In general, it is possible to have N signal groups in an I-PDU and M signal groups protected by E2E, where 0 <= M <= N. In such a case, the callout invokes E2E Library functions M times (for each of the protected signal group).

**Transmitter**

```c
FUNC(boolean, COM_APPL_CODE) IPDU_E2EProtect_21 (PduIdType RxPduId, CONSTP2VAR (PduInfoType, AUTOMATIC, COM_VAR_NOINIT) PduInfoPtr)
{
    /* At first run, instantiate the structures and set the init values */
    static E2E_P01ConfigType Cfg_Write_21 =
        { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
    static E2E_P01ProtectStateType Sta_Write_21 = {0};

    Std_ReturnType ret = E2E_P01Protect(& Cfg_Write_21,
        & Sta_Write_21,
        IPduInfo->SduDataPtr);
    /* return TRUE if no error in protect function */
    return (ret != 0);
}
```

**Receiver**

```c
FUNC(boolean, COM_APPL_CODE) IPDU_E2ECheck_21 (PduIdType RxPduId, CONSTP2CONST (PduInfoType, AUTOMATIC, COM_VAR_NOINIT) PduInfoPtr) {
    /* At first run, instantiate the structures and set the init values */
    static E2E_P01ConfigType Cfg_Read_21 =
        { 64, 21, E2E_P01_DATAID_BOTH, 1, 0, 8 };
    static E2E_P01CheckStateType Sta_Read_21 =
        {0, 0, TRUE, FALSE, E2E_P01STATUS_NONEWDATA};
    /* If callout is invoked, this means that new data is available at COM */
    Sta_Read_21.NewDataAvailable = TRUE;
    Std_ReturnType ret = E2E_P01Check(Cfg_Read_21, Sta_Read_21,
        IPduInfo->SduDataPtr);
    /* return TRUE if no error, possibly only some messages lost Within counter tolerance */
    if( ret == E2E_OK &&
        (Sta_Read_21.Status == E2E_P01STATUS_OK ||
        (Sta_Read_21.Status == E2E_P01STATUS_OK ||
```
12.3 Provision of the Protection Wrapper Interface on a ECU with COM Callout solution

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

12.4 Protection at RTE level through E2E Transformer

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

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

This scenario is specified in details in SWS E2E Transformer.

principles are provided in this chapter.independently.
13 Usage and generation of DataIDLists for E2E profile 2

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

Any receiver of the intended message needs to know the DataID a priori. The performed check of the received CRC at the receiver side does only match if and only if the assumed DataID on the receiver side is identical to the DataID used at the sender side. Thus, the DataID allows protecting messages against masquerading. It is important that the used DataID is known solely by the intended sender and the intended receiver.

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

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

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

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

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

13.1 Example A (persistent routing error)

Assumptions
Consider a network with one or more nodes as sender (messages A to F) and one node as the intended receiver of the safety relevant message (message B). The messages are configured to use the DataIDList as shown in Figure 13-1 and Figure 13-2.
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 \textit{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 \textit{not valid}.

\textbf{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 \textit{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).
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.

<table>
<thead>
<tr>
<th>Receiver-ECU</th>
<th>DataIDList</th>
<th>Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver</td>
<td>G</td>
<td>73 144 215 35 106 177 248 68 139 210 30 101 172 243 63 134</td>
</tr>
</tbody>
</table>

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:

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

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.
Specification of SW-SW Communication Protection Library

- AUTOSAR confidential

30 of 315

For each value of counter DataID: value to be used

- "": not yet assigned
- "X": not allowed
- "": not allowed
For each value of counter: DataID value to be used

For a message with length [bytes]: "": not yet assigned, "X": not allowed
## Specification of SW-C End-to-End Communication Protection Library

### AUTOSAR CP Release 4.3.1

| # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 |
| 0 | **For each value of counter:** DataID value to be used | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 |
| 1 | **For a message with length [bytes]:** "X": not yet assigned, "X": not allowed | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 |

308 of 315

- AUTOSAR confidential -

Document ID 428: AUTOSAR_SWS_E2ELibrary
### Specification of SW-C End-to-End Communication Protection Library

**AUTOSAR CP Release 4.3.1**

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

<table>
<thead>
<tr>
<th>#</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>#</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

309 of 315

- AUTOSAR confidential -

Document ID 428: AUTOSAR_SWS_E2ELibrary
### Specification of SW-C End-to-End Communication Protection Library

**AUTOSAR CP Release 4.3.1**

<table>
<thead>
<tr>
<th>DataID value to be used</th>
<th>For a message with length [bytes]:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;&quot;: not yet assigned, &quot;X&quot;: not allowed</td>
</tr>
<tr>
<td>0</td>
<td>g, i, j, k</td>
</tr>
<tr>
<td>1</td>
<td>i, j, k, m</td>
</tr>
<tr>
<td>2</td>
<td>j, k, m, n</td>
</tr>
<tr>
<td>3</td>
<td>k, m, n, o</td>
</tr>
<tr>
<td>4</td>
<td>m, n, o, p</td>
</tr>
<tr>
<td>5</td>
<td>n, o, p, q</td>
</tr>
<tr>
<td>6</td>
<td>o, p, q, r</td>
</tr>
<tr>
<td>7</td>
<td>p, q, r, s</td>
</tr>
<tr>
<td>8</td>
<td>q, r, s, t</td>
</tr>
<tr>
<td>9</td>
<td>r, s, t, u</td>
</tr>
<tr>
<td>10</td>
<td>s, t, u, v</td>
</tr>
<tr>
<td>11</td>
<td>t, u, v, w</td>
</tr>
<tr>
<td>12</td>
<td>u, v, w, x</td>
</tr>
<tr>
<td>13</td>
<td>v, w, x, y</td>
</tr>
<tr>
<td>14</td>
<td>w, x, y, z</td>
</tr>
<tr>
<td>15</td>
<td>x, y, z, #</td>
</tr>
</tbody>
</table>

**For each value of counter:**

- **DataID value to be used**
  - 0: g, i, j, k
  - 1: i, j, k, m
  - 2: j, k, m, n
  - 3: k, m, n, o
  - 4: m, n, o, p
  - 5: n, o, p, q
  - 6: o, p, q, r
  - 7: p, q, r, s
  - 8: q, r, s, t
  - 9: r, s, t, u
  - 10: s, t, u, v
  - 11: t, u, v, w
  - 12: u, v, w, x
  - 13: v, w, x, y
  - 14: w, x, y, z
  - 15: x, y, z, #
- **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 | 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 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 2 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 3 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 4 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 5 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 6 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 7 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 8 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
| 9 | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX |
For each value of counter: DataID value to be used

```
<table>
<thead>
<tr>
<th>Value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
For a message with length [bytes]: 

- "": not yet assigned
- "X": not allowed
```
| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 100 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 150 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 200 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 250 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 300 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 350 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 400 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 450 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 500 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 550 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 600 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 650 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 700 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 750 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 800 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

For each value of counter: DataID value to be used

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

331 of 315

Document ID 428: AUTOSAR_SWS_E2ELibrary

- AUTOSAR confidential -

Specification of SW-C End-to-End Communication Protection Library

AUTOSAR CP Release 4.3.1
14 Not applicable requirements

[SWS_E2E_NA_00294] These requirements are not applicable to this specification. (SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00007, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00158, SRS_BSW_00160, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00168, SRS_BSW_00172, SRS_BSW_00300, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00304, SRS_BSW_00305, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00310, SRS_BSW_00312, SRS_BSW_00314, SRS_BSW_00318, SRS_BSW_00321, SRS_BSW_00325, SRS_BSW_00327, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00333, SRS_BSW_00334, SRS_BSW_00335, SRS_BSW_00336, SRS_BSW_00338, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00346, SRS_BSW_00347, SRS_BSW_00348, SRS_BSW_00349, SRS_BSW_00350, SRS_BSW_00351, SRS_BSW_00352, SRS_BSW_00353, SRS_BSW_00354, SRS_BSW_00355, SRS_BSW_00356, SRS_BSW_00357, SRS_BSW_00358, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00361, SRS_BSW_00369, SRS_BSW_00371, SRS_BSW_00373, SRS_BSW_00374, SRS_BSW_00375, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00380, SRS_BSW_00381, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00387, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00391, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00394, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00402, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00407, SRS_BSW_00408, SRS_BSW_00409, SRS_BSW_00410, SRS_BSW_00411, SRS_BSW_00412, SRS_BSW_00413, SRS_BSW_00414, SRS_BSW_00415, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00418, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00430, SRS_BSW_00431, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00434, SRS_BSW_00435, SRS_BSW_00436, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00439, SRS_BSW_00440, SRS_BSW_00441, SRS_BSW_00442, SRS_BSW_00443, SRS_BSW_00444, SRS_BSW_00445, SRS_BSW_00446, SRS_BSW_00447, SRS_BSW_00448, SRS_BSW_00449, SRS_BSW_00450, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00453, SRS_BSW_00454, SRS_BSW_00455, SRS_BSW_00456, SRS_BSW_00457, SRS_BSW_00458, SRS_BSW_00459, SRS_BSW_00460, SRS_BSW_00461, SRS_BSW_00462, SRS_BSW_00463, SRS_BSW_00464, SRS_BSW_00465, SRS_BSW_00466, SRS_BSW_00467, SRS_BSW_00468, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00473, SRS_BSW_00474, SRS_BSW_00475, SRS_BSW_00476, SRS_BSW_00477, SRS_BSW_00478, SRS_BSW_00479, SRS_BSW_00480, SRS_BSW_00481, SRS_BSW_00482, SRS_BSW_00483, SRS_E2E_08535, SRS_LIBS_00001, SRS_LIBS_00002, SRS_LIBS_00003, SRS_LIBS_00004, SRS_LIBS_00005, SRS_LIBS_00007, SRS_LIBS_00008, SRS_LIBS_00009, SRS_LIBS_00010, SRS_LIBS_00011, SRS_LIBS_00012, SRS_LIBS_00013, SRS_LIBS_00015, SRS_LIBS_00016, SRS_LIBS_00017, SRS_LIBS_00018, SRS_LIBS_08518, SRS_LIBS_08521, SRS_LIBS_08525, SRS_LIBS_08526)