AUTOSAR E2E User Manual

Document Change History				
Date (YYYY-MM-DD)	Ver.	Editor	Chap	Description
2020-03-12	1.0.0	CuongLX	All	• E2E Module Manual Initial Creation
2021-12-29	1.0.1.0	HiepVT1	4.2 4.3	• Scope of Release • Update 4.3. Change Log • Applying change of company name
2022-06-30	1.0.1.1	Gongbin Lim	1 4.3	• Clarify copyright of source code • Update 4.3 Change Log
2022-08-12	1.0.2.0	Gongbin Lim	4.3	• Fix UNECE Polyspace
2022-08-29	1.0.2.1	Gongbin Lim	4.3	• Update 4.3 Change Log
2023-10-11	1.0.2.2	Seungjin Noh	4.3	• Update 4.3 Change Log

Table of Contents

1. Overview

This document provides the user with caution or reference when setting parameters or designing the system when using the End-To-End Protection function that satisfies the AUTOSAR standard on the AUTOSAR platform.
Please refer to the Reference document for details.
The interpretation of the category related to setting is as follows.

- Changeable(C): Items that can be set by the user
- Fixed(F): Items that cannot be changed by the user
- Not Supported(N): Deprecated item

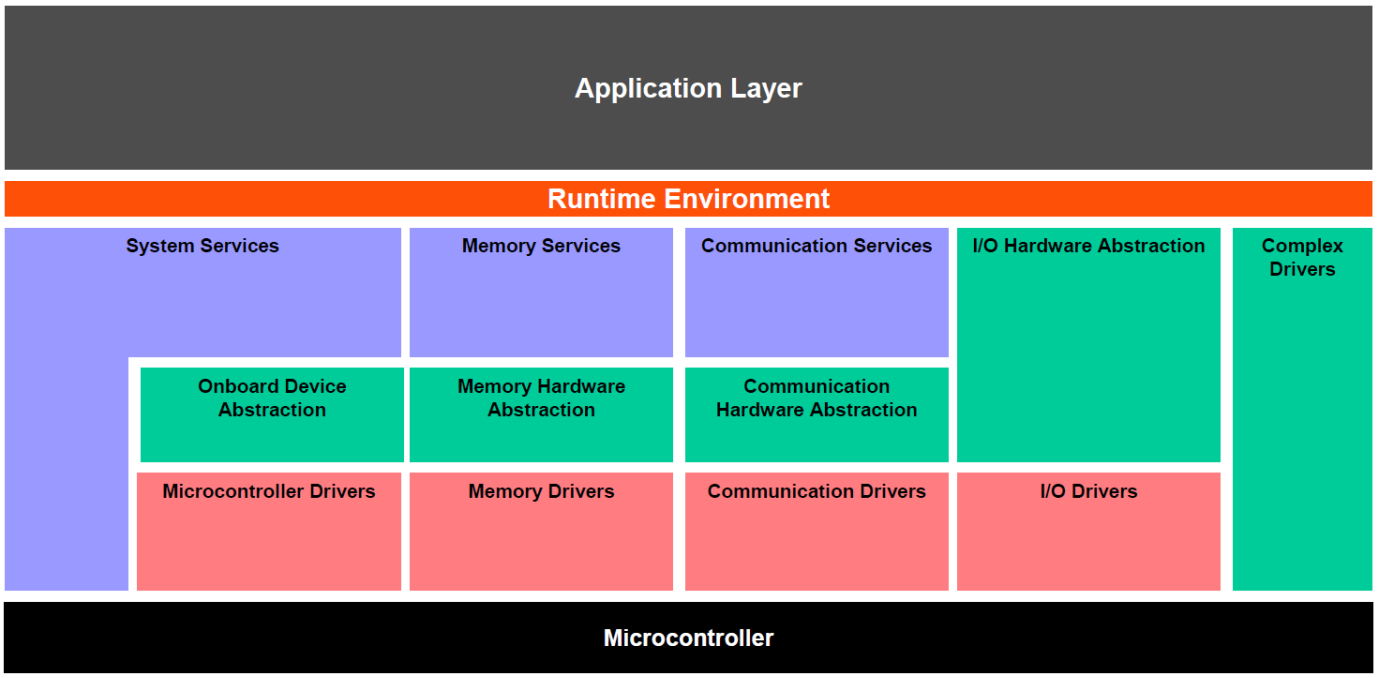
2. Reference

Sl. No.	Title	Version
1.	Requirements on Libraries (AUTOSAR_SRS_Libraries.pdf)	4.4.0
2.	Specification of SW-C End-to-End Communication Protection Library (AUTOSAR_SWS_E2ELibrary.pdf)	4.4.0
3	E2E Protocol Specification (AUTOSAR_PRS_E2EProtocol.pdf)	1.5.1
4	Requirements on E2E Communication Protection (AUTOSAR_SRS_E2E.pdf)	4.4.0

3. AUTOSAR System

3.1 Overview of Software Layers

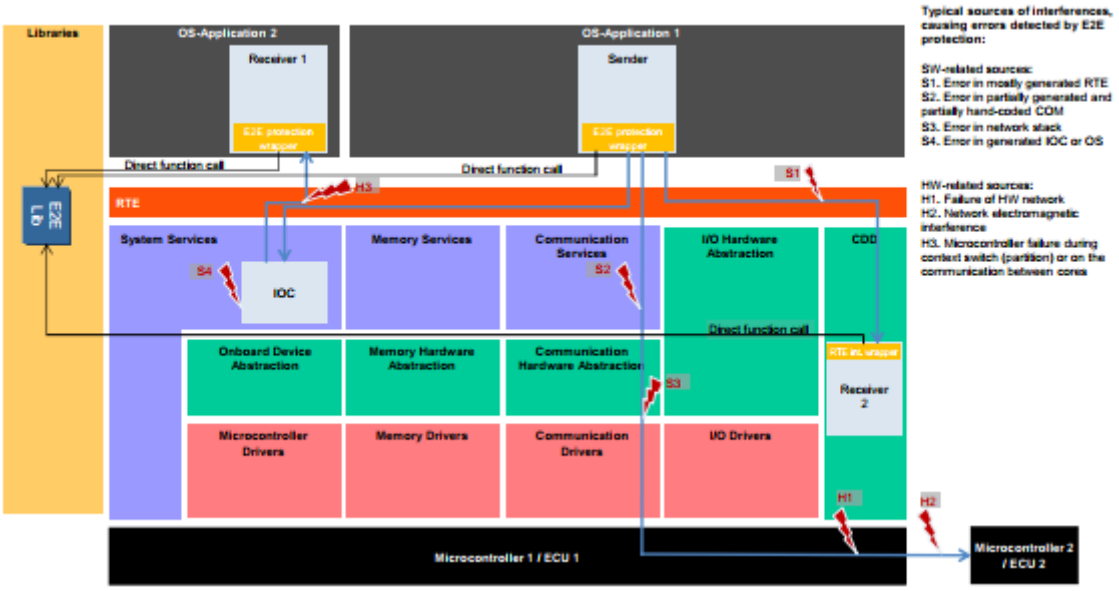
Layered Architecture of AUTOSAR Platform is as following. The AUTOSAR platform can be divided into Service Layer, ECU Abstraction Layer, Complex Device Drivers, and Microcontroller Abstraction Layer.



3.2 AUTOSAR E2E Module

E2E starts from the concept of protecting data from the effects of the following faults during data transmission and reception during the execution of the controller. Examples of such defects are:

- HW Faults (e.g. register errors in CAN transceivers)
- Radio interference (e.g. EMC)
- System errors in software implementing VFB (e.g. RTE, IOC, COM and network stack)



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 suitable safety concepts, e.g. to achieve freedom from interference between system elements sharing a common communication infrastructure (see ISO 26262-6:2011, annex D.2.4):

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

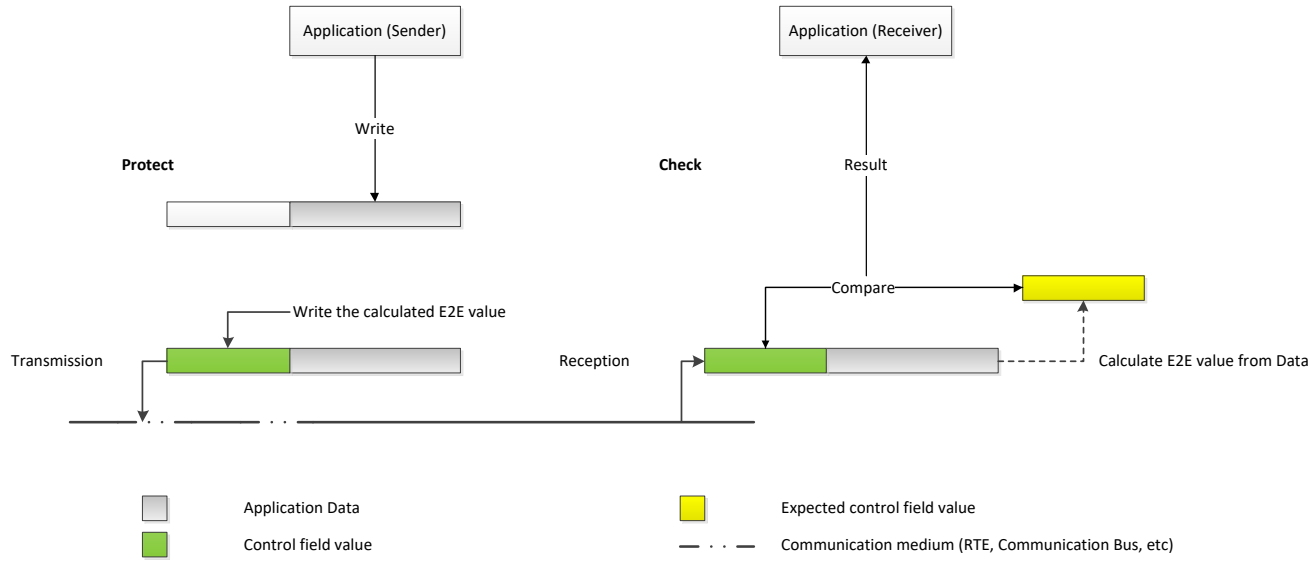
The E2E communication protection allows the following:

1. It protects the safety-related data to be sent by adding control data,
2. It verifies the safety-related data received using this control data, and
3. It provides the check result to the receiver, which then has to handle it sufficiently.

3.2.1 E2E Library

The E2E Library provides the following functions.
Sender: Updates control fields such as CRC and Counter to the E2E Header area in the transmitted data (Protect)
Receiver: Evaluate the E2E Header of the received data (Check)

Receiver: (Calculate the expected control field value for the received data and compare it with the received value)



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

1. A CRC, provided by CRC Supervision;
2. A Sequence Counter incremented at every transmission request, the value is checked at receiver side for correct increment;
3. An Alive Counter incremented at every transmission request, the value checked at the receiver side if it changes at all, but correct increment 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:

(a) Receiver communication timeout.

(b) Sender acknowledgement timeout.

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

3.2.1.1 E2E Profile 1

The functions of E2E Profile 1 suggested in AUTOSAR specification are as follows.

E2E Mechanism	Detected communication faults
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Transmission on a regular basis and timeout monitoring using E2E-Supervision [1]	Loss, delay, blocking
Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, Asymmetric information [2]

In the E2E Profile 1, the layout is in general free to be defined by the user, as long as the basic limitations of signal alignment are followed:

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

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

Counter: The counter area is 4 bits, and the position of the counter can be specified by the user. Counter: The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 14.

Data ID: Data ID is 16 bits, and Data ID is not directly transmitted and is used for CRC calculation. (See picture)

Data ID: User can specify how to use Data ID in CRC calculation through Data ID Mode.

Data ID: There are the following types of Data ID Mode.

- E2E_P01_DATAID_BOTH: both two bytes (double ID configuration) are included in the CRC, first low byte and then high byte
- E2E_P01_DATAID_ALT: depending on parity of the counter (alternating ID configuration) the high and the low byte is included. For even counter values the low byte is included and for odd counter values the high byte is included.
- 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.
- 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,

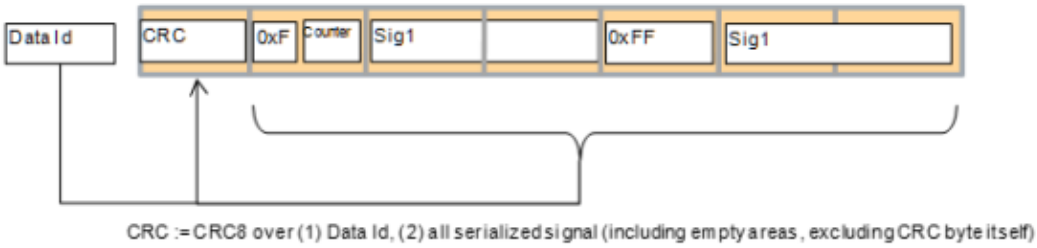
- The low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.
- the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for the inclusion modes _BOTH, _ALT and _LOW)

CRC: The CRC area is 8 bits, and the CRC location can be specified by the user. CRC: When calculating CRC, use the CRC_CalculateCRC8 () API of the CRC module.

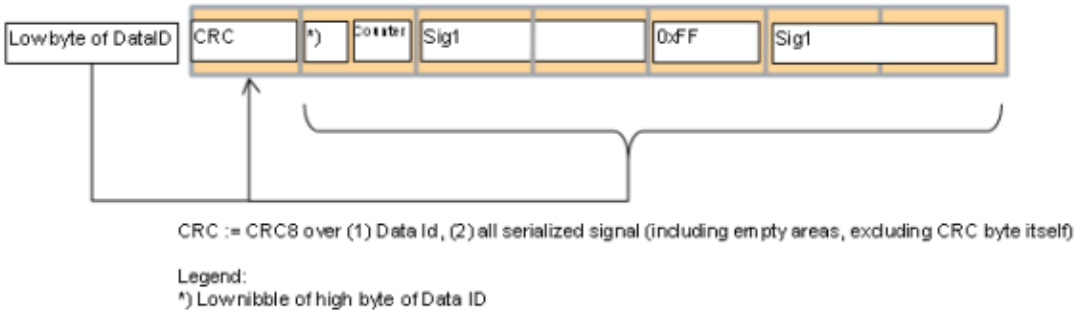
Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

Variant exists in E2E Profile1 and is as follows.

- Variant 1A:



- Variant 1C:



3.2.1.2 E2E Profile 2

The functions of E2E Profile 2 suggested in AUTOSAR specification are as follows.

E2E Mechanism	Detected communication faults
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Transmission on a regular bases and timeout monitoring using E2E-Library [3]	Loss, delay, blocking
Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, Asymmetric information [4]

The data layout of E2E Profile 2 can be expressed as follows. (Example)

Counter: Counter area is 4 bits, and the position of the counter is fixed to the 8th bit. The Counter

value increases by 1 for each Protect request from the Sender, and the range is 0 to 15.

Data ID: Data ID is 8 bits, and a total of 16 Data ID sets are used. Data ID is used for CRC calculation without direct transmission. CRC is calculated using the Data ID of the index corresponding to the current Counter value. (For example, if the current Counter is 8, CRC is calculated using the 8th Data ID.)

CRC: The CRC area is 8 bits, and the CRC position is fixed to the 0th bit. When calculating CRC, use the CRC_CalculateCRC8H2F () API of the CRC module.

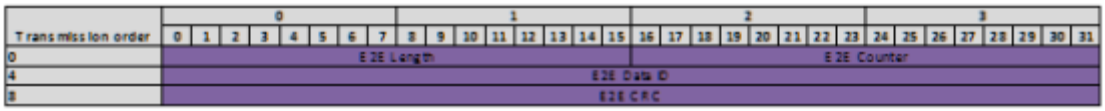
Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

3.2.1.3 E2E Profile 4

The functions of E2E Profile 4 suggested in AUTOSAR specification are as follows.

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

The data layout of E2E Profile 4 can be expressed as follows. (Example)



Counter: Counter area is 16 bits, and the position of the counter is fixed from 16th to the 31th bit. The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 0xFFFF.

Data ID: Data ID is 32 bits, and the Data ID shall be explicitly transmitted i.e. it shall be the part of the transmitted E2E header.

CRC: The CRC area is 32 bits, and the CRC position is fixed to the 64th bit. When calculating CRC, use the CRC_CalculateCRC32P4 () API of the CRC module.

Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

3.2.1.4 E2E Profile 5

The functions of E2E Profile 4 suggested in AUTOSAR specification are as follows.

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

The header of E2E Profile 5 (the location of the CRC and Counter) can be expressed as follows.

	0								1								2							
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
0	E2E CRC																E2E Counter							

Counter: Counter area is 8 bits and is located after CRC in the header. The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 0xFF.

Data ID: Data ID is 16 bits and has uniqueness throughout the system. Data ID is not directly transmitted and is used for CRC calculation.

CRC: CRC area is 16 bits (polynomial in normal form 0x1021 (AUTOSAR notation)), the CRC position is the first 2 bytes of the header. When calculating CRC, use the CRC_CalculateCRC16 () API of the CRC module.

Length: Length is not transmitted directly.

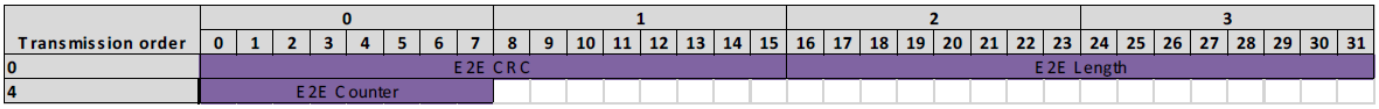
Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

3.2.1.5 E2E Profile 6

The functions of E2E Profile 6 suggested in AUTOSAR specification are as follows.

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

The header (CRC, Length and Counter position) of E2E Profile 6 can be expressed as follows.



Counter: Counter area is 8 bits and is located after Length in the header. The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 0xFF.

Data ID: Data ID is 16 bits and has uniqueness throughout the system. Data ID is not directly transmitted and is used for CRC calculation.

CRC: CRC area is 16 bits (polynomial in normal form 0x1021 (AUTOSAR notation)), the CRC position is the first 2 bytes of the header. When calculating CRC, use the CRC_CalculateCRC16 () API of the CRC module.

Length: The length area is 16 bits and is located after the CRC in the header. Indicates the length of data to be transmitted.

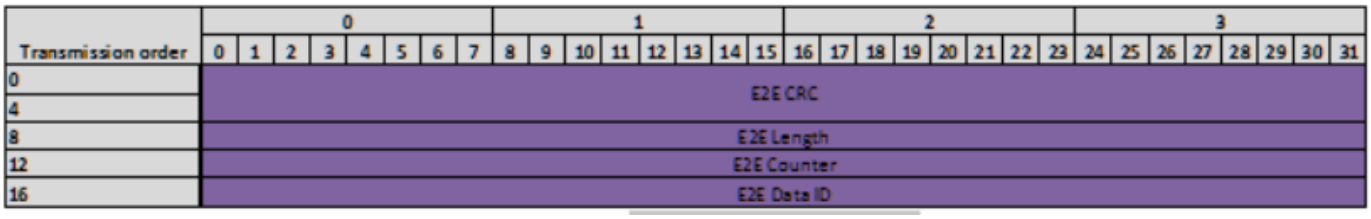
Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

3.2.1.6 E2E Profile 7

The functions of E2E Profile 7 suggested in AUTOSAR specification are as follows.

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

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



Counter: Counter area is 32 bits and is located after Length in the header. The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 0xFF'FF'FF'FF.

Data ID: Data ID is 32 bits and has uniqueness throughout the system. The Data ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.

CRC: CRC area is 64, the CRC position is the first 4 bytes of the header. When calculating CRC, use the CRC_CalculateCRC64() API of the CRC module.

Length: The length area is 32 bits and is located after the CRC in the header. Indicates the length of data to be transmitted.

Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

3.2.1.7 E2E Profile 11

The functions of E2E Profile 11 suggested in AUTOSAR specification are as follows.

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

The header of the E2E Profile 11 (the location of the CRC and Counter) can be expressed as follows.

Byte Order	0								1							
Transmission Order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit Order	7	6	5	4	3	2	1	0	15	14	12	12	11	10	9	8
	E2E CRC								DataIDNibble				Counter			

Option

Counter: The counter area is 4 bits, and the position of the counter can be specified by the user. The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 14.

Data ID: Data ID is 16 bits, and Data ID is used for CRC calculation. User can specify how to use Data ID in CRC calculation through Data ID Mode. There are the following types of Data ID Mode.

1. E2E_P11_DATAID_BOTH: Data ID of 2 bytes is divided into upper / lower bytes and used for CRC calculation.
2. E2E_P11_DATAID_NIBBLE: The lower byte of the Data ID is used for CRC calculation, and the lower nibble of the upper byte is included in the header and transmitted (user can specify the location).

CRC: The CRC area is 8 bits, and the CRC location can be specified by the user. When calculating CRC, use the CRC_CalculateCRC8 () API of the CRC module.

Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

Variant exists in E2E Profile11 and is as follows.

- Variant 11C: CRC position is 0th bit, Counter position is 8th bit, Data ID Nibble position is fixed at 12th bit,
- Variant 1A: Data ID uses both the upper byte nibble (for header inclusion) and lower byte (for CRC calculation). (E2E_P11_DATAID_NIBBLE). The CRC position is fixed at the 0th bit and the counter position at the 8th bit,
- Variant 1B: Data ID uses high / low byte alternately. (E2E_P01_DATAID_ALT)

3.2.1.8 E2E Profile 22

The functions of E2E Profile 22 suggested in AUTOSAR specification are as follows.

E2E Mechanism	Detected communication faults
Counter	Repetition, loss, insertion, incorrect sequence, blocking
Transmission on a regular bases and timeout monitoring using E2E-Library [5]	Loss, delay, blocking
Data ID + CRC	Masquerade and icorrect addressing, insertion
CRC	Corruption, asymmetric information [6]

The header of the E2E Profile 11 (the location of the CRC and Counter) can be expressed as follows.

Byte Order	0								1							
Transmission Order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit Order	7	6	5	4	3	2	1	0	15	14	12	12	11	10	9	8
0	E2E CRC												Counter			

Counter: Counter area is 8 bits, and the position of the counter is fixed to the 0th bit. The Counter value increases by 1 for each Protect request from the Sender, and the range is 0 to 15.

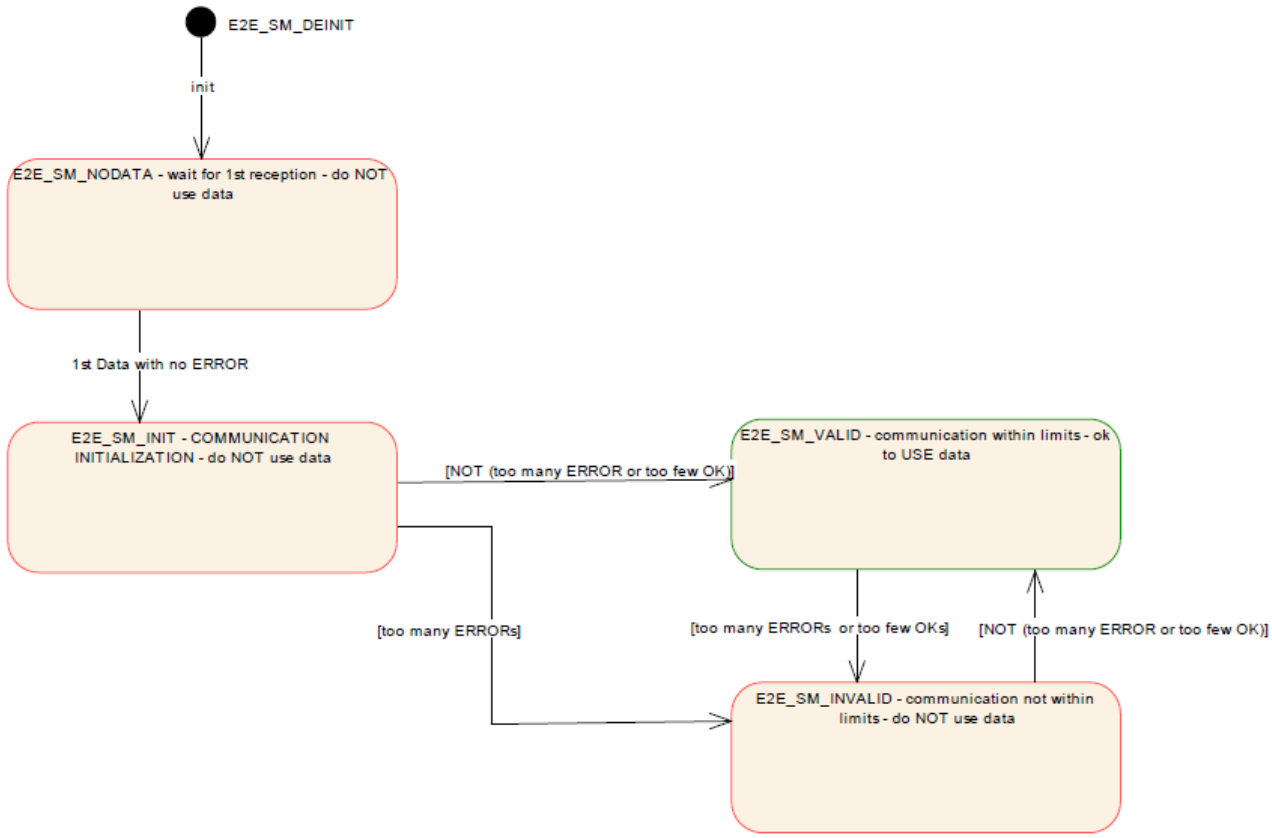
Data ID: Data ID is 16 bits and has uniqueness throughout the system. Data ID is not directly transmitted and is used for CRC calculation.

CRC: The CRC area is 8 bits, and the CRC position is fixed to the 0th bit. When calculating CRC, use the CRC_CalculateCRC8H2F () API of the CRC module.

Timeout: When checking in Receiver, availability of new data and check whether the data is duplicated through the counter, and check if a timeout occurs.

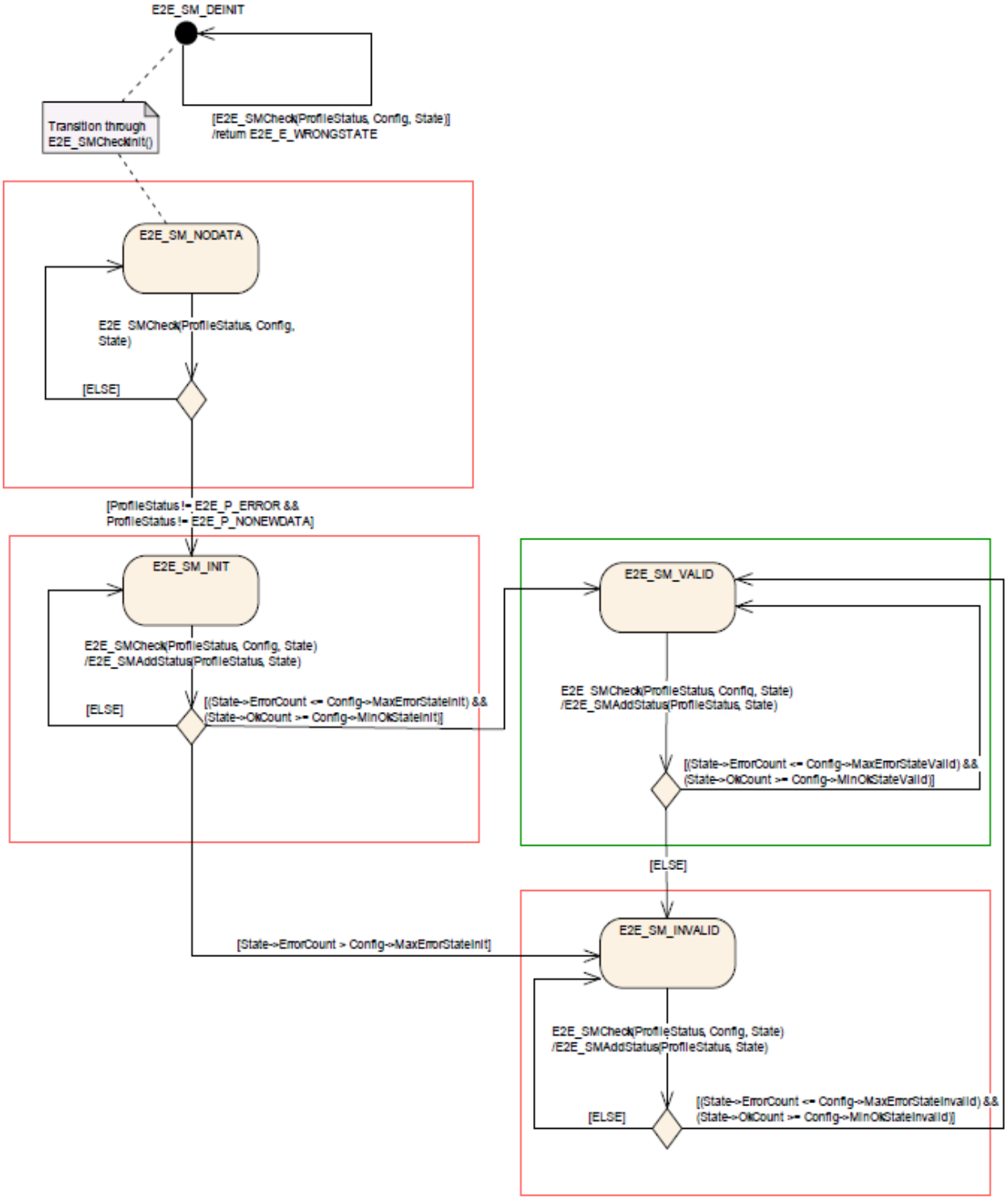
3.2.1.9 E2E State Machine

The E2E Profile Check function verifies data for each cycle. The E2E State Machine stores the result values of the Check function and provides them to RTE / SWC / COM.
E2E State Machine is applicable to all E2E Profiles.
The diagram of E2E State Machine can be summarized as follows.



- It consists of 5 states: E2E_SM_DEINIT, E2E_SM_NODATA, E2E_SM_INIT, E2E_SM_VALID, and E2E_SM_INVALID.
- Depending on the results of the E2E Profile Check function, each state changes or maintains state.

E2E State Machine is composed of E2E_SMCheckInit () and E2E_SMCheck () functions. E2E_SMCheck function



- In the case of E2E_SM_DEINIT state, if the E2E_SM_Check function is called, it exits with the result value of E2E_E_WRONGSTATE and maintains the state.
- When the E2E_SM_CheckInit function is called, it switches from E2E_SM_DEINIT state to E2E_SM_NODATA state and ends the function.
- In case of E2E_SM_NODATA status, if E2E_SM_Check function is called, if the Status of the Profile received as the argument (the most recent E2E Profile Check function result status) is not E2E_P_ERROR and E2E_P_NONEWDATA, the function is switched to E2E_SM_INIT state and the function

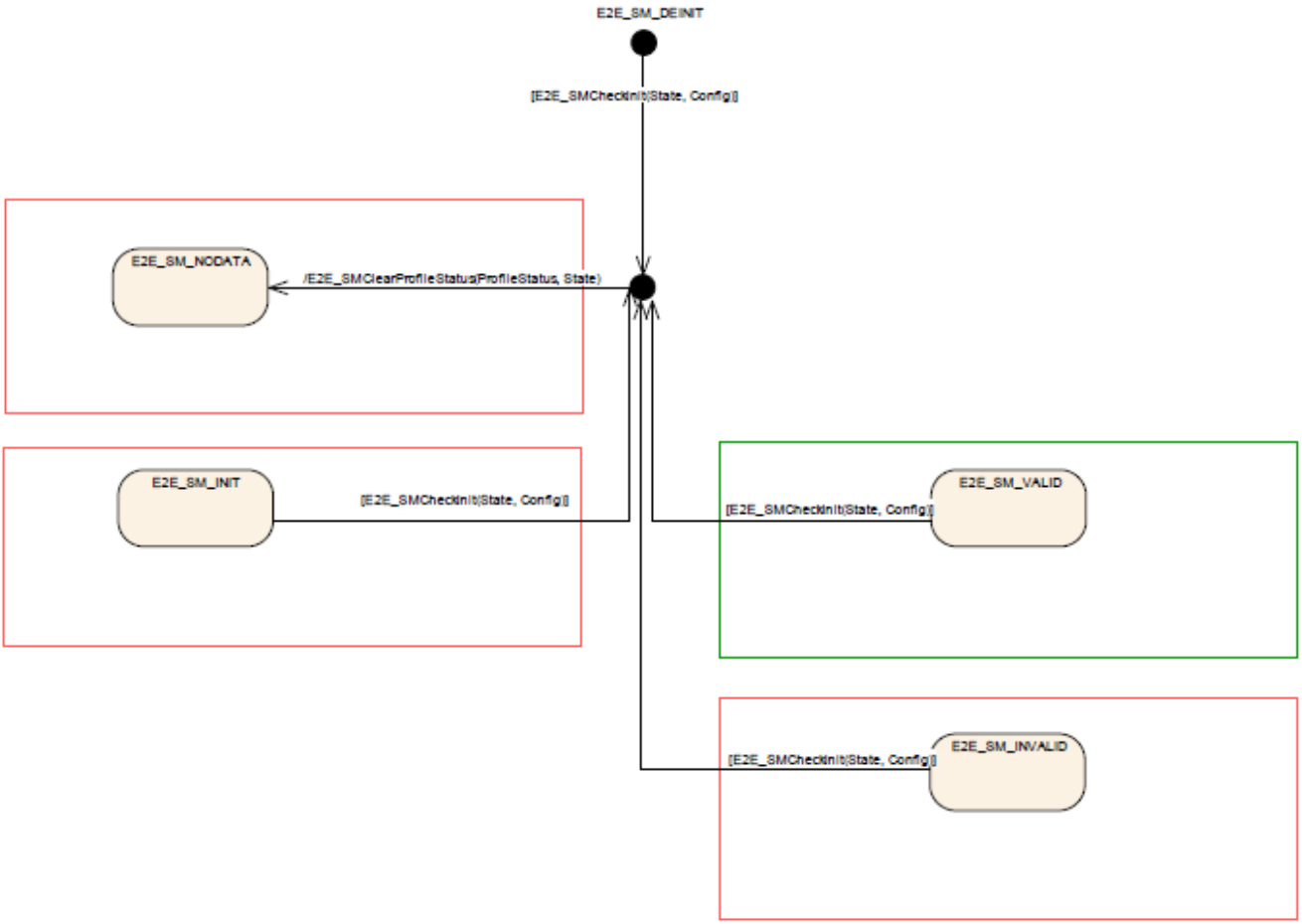
is terminated.

- If ProfileStatus is E2E_P_ERROR or E2E_P_NONEWDATA, maintain the status and exit the function.
- In case of E2E_SM_INIT state, when the E2E_SM_Check function is called, the E2E_SMAddStatus function is executed to update the result of the most recent E2E Profile Check function.
- After the update is completed, compare the preset number of ERRORs with the number of OKs to switch the state to E2E_SM_VALID, E2E_SM_INVALID, or maintain the state and exit the function. If E2E_SM_Check function is called in E2E_SM_VALID, E2E_SM_INVALID state, it operates in a similar way as above.

The E2E_SMAddStatus function operates as follows.

- Updates the ProfileStatus (result status of the recently operated E2E Profile Check function) received as an argument.
- E2E_P_OK is updated by scanning the updated ProfileStatus storage array.
- E2E_P_ERROR is updated by scanning the updated ProfileStatus storage array.
- Increase Index of ProfileStatus storage array by 1.

The diagram of E2E_CheckInit function is as follows.



- Regardless of the state of the state machine, after executing the E2E_SMClearStatus function, it switches to the E2E_SM_NODATA state.

The E2E_SMClearStatus function operates as follows.

- Initialize the elements of the ProfileStatus storage array to E2E_P_NOTAVAILABLE.
- Initialize the E2E_P_OK count and E2E_P_ERROR count to 0.
- Initialize the Index of ProfileStatus storage array to 0.
- Therefore, it should be set so that the ProfileStatus storage array does not become a NULL pointer (in the case of a NULL pointer, E2E_E_INPUTERR_NULL occurs)

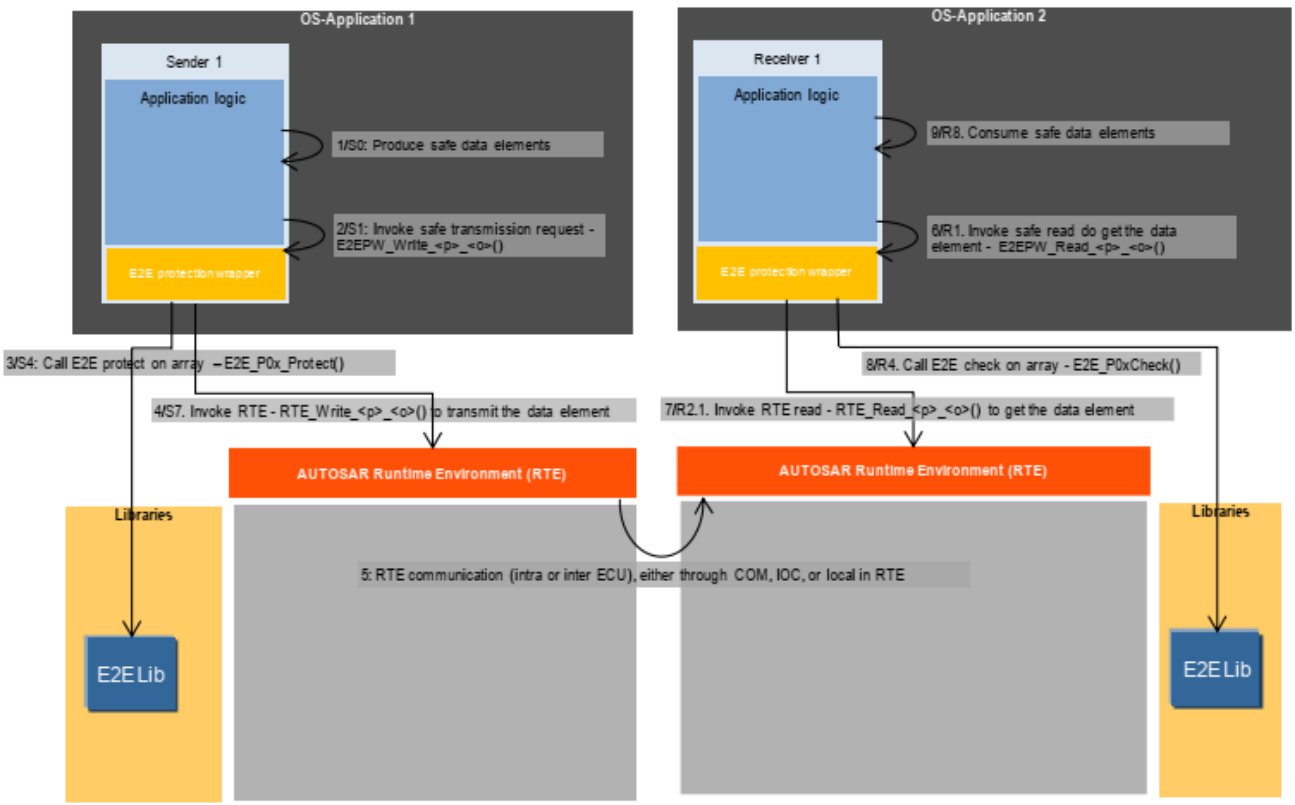
3.2.2 Usage of E2E Library

The E2E library is invoked from:

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

3.2.2.1 E2E Protection Wrapper

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.
E2E Protection Wrapper works as follows.



Rte_Write API is called to send application data including the E2E header to Rte.

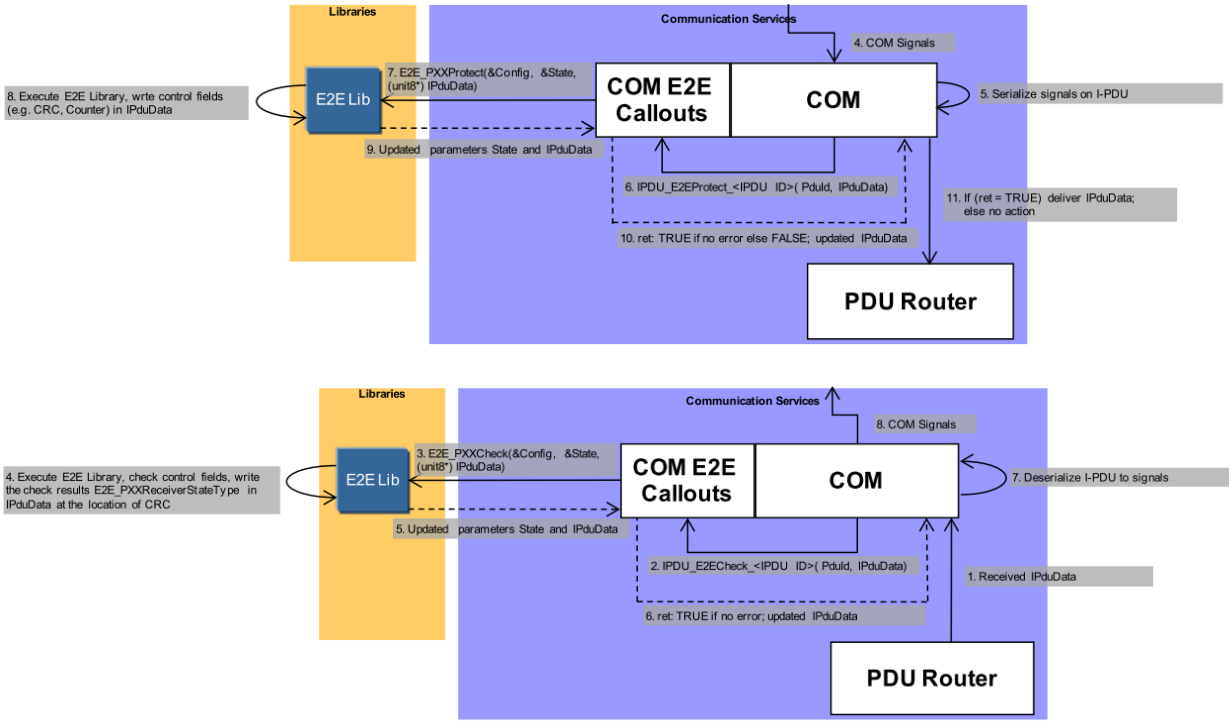
- For Receiver, after receiving data from Rte by calling Rte_Read,
- After calling the E2E library to determine whether the data is normal through the E2E header,

- The result and the received data are returned.

E2E Protection Wrapper
It can be used for both SW-C communication through RTE and controller communication via CAN in the controller.
In the case of communication between controllers, a function of arranging data in the same layout as the I-PDU may be added. (Serialization)
It also provides redundant function that compares the results by performing Protect and Check twice to secure reliability.

3.2.2.2 COM E2E Callouts

COM E2E Callouts use Com's I-PDU Callout function, Com protects data before sending I-PDU to PduR,
After receiving the I-PDU from the PduR, this function determines whether to use the data by examining the data.
COM E2E Callouts works as follows.



For Sender,
Call COM E2E Callouts registered in I-PDU Callout before sending I-PDU from Com to PduR.
COM E2E Callouts protect E2E by calling E2E library for Signal Group of I-PDU.
Com sends an I-PDU with E2E header to PduR.
Receiver
When Com receives I-PDU from PduR, it calls COM E2E Callouts registered in I-PDU Callout.
COM E2E Callouts call E2E library for I-PDU Signal Group
It is determined whether the data is normal, and the result is returned.
Com takes the I-PDU if the result is true, and discards the I-PDU if it is False.
COM E2E Callouts can only be used for control period communication using the Com module.
Defects due to errors of RTE and Com cannot be detected.

3.2.2.3 E2E Transformer

E2E Transformer currently supports E2E Profile 5,6,11. In addition, E2E Transformer uses E2E State Machine.
For E2E Transformer, refer to the E2E Transformer User Manual.

4. Product Release Notes

4.1 Overview

In Chapter, it aims to provide Release-related contents for Hyundai AutoEver E2E Stack, and describes limitations and specifics for E2E Stack Software product release version.

4.2 Scope of the release

All information in this document is limited to the following Hyundai AutoEver E2E modules:

Module	AUTOSAR version	Module version
E2E	4.4.0	1.0.2

※ Module version means Sw version of BswModuleDescription (Bswmd) file of each module.

4.3 Change Log

4.3.1 Version 1.0.2.2 (2023-10-11)

- Task
  - Update polyspace document

4.3.2 Version 1.0.2.1 (2022-08-29)

- Task
  - Add new requirement for SAD

4.3.3 Version 1.0.2.0 (2022-08-12)

- Task
  - Fix UNECE Polyspace

4.3.4 Version 1.0.1.1 (2022-06-30)

- Improvement
  - Clarify the copyright of code in E-Code

- Divide 'delivery' folder into 'delivery/src' and 'delivery/inc'
- Apply the latest template of DeliveryBoxHistory

4.3.5 Version 1.0.1.0 (2020-12-29)

- Change Request
  - Change company naming convention, naming files
- Change Request
  - Update some work products for ASPICE tasks
- Defect
  - Fix INT31-C violation for all of Profiles

4.3.6 Version 1.0.0.0 (2020-03-06).

- New version

4.4 Limitations

NONE.

4.5 Deviations

NONE.

5. Configuration Guide

5.1 E2E

None

5.2 System Configuration

None

6. Application Programming Interface (API)

6.1 Type Definitions

6.1.1 E2E Profile 1 types

E2E_P01ConfigType

Name:	E2E_P01ConfigType		
Type:	Structure		
Element:	uint16	CounterOffset	Bit offset of Counter in MSB first order. CounterOffset shall be a multiple of 4. In variants 1A, 1B, and 1C, CounterOffset is 8.
	uint16	CRCOffset	Bit offset of CRC (i.e. since *Data) in MSB first order. The offset shall be a multiple of 8. In variants 1A, 1B, and 1C, CRCOffset is 0.
	uint16	DataID	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section "Configuration constraints on Data IDs".
	uint16	DataIDNibbleOffset	Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P01_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than E2E_P01_DATAID_NIBBLE, DataIDNibbleOffset shall be initialized to 0 (even if it is ignored by E2E Library).
	E2E_P01DataIDMode	DataIDMode	Inclusion mode of ID in CRC computation (both bytes, alternating, or low byte only of ID included).
	uint16	DataLength	Length of data, in bits. The value shall be a multiple of 8 and shall be ≤ 240.

	uint8	MaxDeltaCounterInit	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	uint8	MaxNoNewOrRepeatedData	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	uint8	SyncCounterInit	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
Description:	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 1. For each transmitted Data, there is an instance of this typedef.		
Available via:	E2E.h		

E2E_P01DataIDMode

Name:	E2E_P01DataIDMode		
Type:	Enumeration		
Range:	E2E_P01_DATAID_BOTH	0	Two bytes are included in the CRC (double ID configuration) This is used in E2E variant 1A.
	E2E_P01_DATAID_ALT	1	One of the two bytes byte is included, alternating high and low byte, depending on parity of the counter alternating ID configuration). For an even counter, the low byte is included. For an odd counter, the high byte is included. This is used in E2E variant 1B.
	E2E_P01_DATAID_LOW	2	Only the low byte is included, the high byte is never used. This is applicable if the IDs in a particular system are 8 bits.
	E2E_P01_DATAID_NIBBLE	3	The low byte is included in the implicit CRC calculation, the low nibble of the high byte is transmitted along with the data (i.e. it is explicitly included), the high nibble of the high byte is not used. This is applicable for the IDs up to 12 bits. This is used in E2E variant 1C.
Description:	The Data ID is two bytes long in E2E Profile 1. There are four inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.		
Available via:	E2E.h		

E2E_P01CheckStateType

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

			E2E_P01STATUS_OKSOMELOST. For other values of Status, the value of LostData is undefined. E2E_P01CheckStatusType Status Result of the verification of the Data, determined by the Check function.
	E2E_P01CheckStatusType	Status	Result of the verification of the Data, determined by the Check function.
	uint8	SyncCounter	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
	uint8	NoNewOrRepeatedDataCounter	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.
Description:	State of the receiver for a Data protected with E2E Profile 1.		
Available via:	E2E.h		

E2E_P01CheckStatusType

Name:	E2E_P01CheckStatusType		
Type:	Enumeration		
Range:	E2E_P01STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P01STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P01STATUS_WRONGCRC	0x02	Error: The data has been received according to communication medium, but 1. the CRC is incorrect (applicable for all E2E Profile 1 configurations) or 2. The low nibble of the high byte of Data ID is incorrect (applicable only for E2E Profile 1 with E2E_P01DataIDMode = E2E_P01_DATAID_NIBBLE). The two above errors can be a result of corruption, incorrect addressing or masquerade.
	E2E_P01STATUS_SYNC	0x03	NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
	E2E_P01STATUS_INITIAL	0x04	Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.
	E2E_P01STATUS_REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
	E2E_P01STATUS_OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.

	E2E_P01STATUS_WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the
			most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description:	Result of the verification of the Data in E2E Profile 1, determined by the Check function.		
Available via:	E2E.h		

6.1.2 E2E Profile 2 types

E2E_P02ConfigType

Name:	E2E_P02ConfigType		
Type:	Structure		
Element:	uint16	DataLength	Length of Data, in bits. The value shall be a multiple of 8.
	uint8[16]	DataIDList	An array of appropriately chosen Data IDs for protection against masquerading.
	uint8	MaxDeltaCounterInit	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	uint8	MaxNoNewOrRepeatedData	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	uint8	SyncCounterInit	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
	uint16	Offset	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.		
Available via:	E2E.h		

P02ProtectStateType

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

E2E_P02CheckStateType

Name:	E2E_P02CheckStateType		
Type:	Structure		
Element:	uint8	LastValidCounter	Counter of last valid received message.
	uint8	MaxDeltaCounter	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
	boolean	WaitForFirstData	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	boolean	NewDataAvailable	Indicates to E2E Library that a new data is available for Library to be checked. This attribute is set by the E2E Library caller, and not by the E2E Library.
	uint8	LostData	Number of data (messages) lost since reception of last valid one.
	E2E_P02CheckStatusType	Status	Result of the verification of the Data, determined by the Check function.
	uint8	SyncCounter	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock- in range) after the detection of an unexpected behavior of a received counter.

	uint8	NoNewOrRepeatedDataCounter	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.
Description:	State of the sender for a Data protected with E2E Profile 2.		
Available via:	E2E.h		

E2E_P02CheckStatusType

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

	E2E_P02STATUS_OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter =MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P02STATUS_WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description:	Result of the verification of the Data in E2E Profile 2, determined by the Check function.		
Available via:	E2E.h		

6.1.3 E2E Profile 4 types

E2E_P04ConfigType

Name:	E2E_P04ConfigType		
Type:	Structure		
Element:	uint32	DataID	A system-unique identifier of the Data.
	uint16	Offset	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 ≤ Offset ≤ MaxDataLength- (12*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	uint16	MinDataLength	Minimal length of Data, in bits. E2E checks that Length is ≥ MinDataLength. The value shall be ≤ 4096*8 (4kB) and shall be ≥ 12*8
	uint16	MaxDataLength	Maximal length of Data, in bits. E2E checks that DataLength is ≤ MinDataLength. The value shall be ≤ 4096*8 (4kB) and it shall be ≥ MinDataLength.

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

E2E_P04ProtectStateType

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

E2E_P04CheckStateType

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

E2E_P04CheckStatusType

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

	E2E_P04STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P04STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description:	Status of the reception on one single Data in one cycle, protected with E2E Profile 4.		
Available via:	E2E.h		

6.1.4 E2E Profile 5 types

E2E_P05ConfigType

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

E2E_P05ProtectStateType

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.		
Available via:	E2E.h		

E2E_P05CheckStateType

Name:	E2E_P05CheckStateType
Type:	Structure

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

E2E_P05CheckStatusType

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

6.1.5 E2E Profile 6 types

E2E_P06ConfigType

Name:	E2E_P06ConfigType		
Type:	Structure		
Element:	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. E2E checks that Length is => MinDataLength. The value shall be ≤ 4096*8 (4kB) and shall be => 5*8.
	uint16	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.		
Available via:	E2E.h		

E2E_P06ProtectStateType

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

E2E_P06CheckStateType

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

E2E_P06CheckStatusType

Name:	E2E_P06CheckStatusType		
Type:	--		
Range:	E2E_P06STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P06STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P06STATUS_REPEATED.
	E2E_P06STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong CRC, wrong length).

	E2E_P06STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P06STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P06STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description:	Status of the reception on one single Data in one cycle, protected with E2E Profile 6.		
Available via:	E2E.h		

6.1.6 E2E Profile 7 types

E2E_P07ConfigType

Name:	E2E_P07ConfigType		
Type:	Structure		
Element:	uint32	DataID	A system-unique identifier of the Data.
	uint32	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- (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.
	uint32	MinDataLength	Minimal length of Data, in bits. E2E checks that Length is ≥ MinDataLength. The value shall be ≥ 20*8 and ≤ MaxDataLength.
	uint32	MaxDataLength	Maximal length of Data, in bits. E2E checks that DataLength is ≤ MinDataLength. The value shall be ≥ MinDataLength
	uint32	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 7. For each transmitted Data, there is an instance of this typedef.		
Available via:	E2E.h		

E2E_P07ProtectStateType

Name:	E2E_P07ProtectStateType
Type:	Structure

Element:	uint32	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_P07Protect() is called, it increments the counter up to 0xFF'FF'FF'FF.
Description:	State of the sender for a Data protected with E2E Profile 7.		
Available via:	E2E.h		

E2E_P07CheckStateType

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

E2E_P07CheckStatusType

Name:	E2E_P07CheckStatusType		
Type:	--		
Range:	E2E_P07STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P07STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P07STATUS_REPEATED.
	E2E_P07STATUS_ERROR	0x02	Error: error not related to counters occurred (e.g. wrong CRC, wrong length, wrong options, wrong Data ID).
	E2E_P07STATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P07STATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P07STATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description:	Status of the reception on one single Data in one cycle, protected with E2E Profile 7.		
Available via:	E2E.h		

6.1.7 E2E Profile 11 types

E2E_P11ConfigType

Name:	E2E_P11ConfigType

Type:	Structure		
Element:	uint16	DataLength	Length of data, in bits. The value shall be a multiple of 8 and shall be ≤ 240.
	uint16	DataID	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section "Configuration constraints on Data IDs".
	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.
	E2E_P11DataIDMode	DataIDMode	--
	uint16	CRCOffset	Bit offset of CRC (i.e. since *Data) in MSB first order. In variants 1A and 1B, CRCOffset is 0. The offset shall be a multiple of 8.
	uint16	CounterOffset	Bit offset of Counter in MSB first order. In variants 1A and 1B, CounterOffset is 8. The offset shall be a multiple of 4.
	uint16	DataIDNibbleOffset	Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = 2E_P01_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than E2E_P01_DATAID_NIBBLE, DataIDNibbleOffset shall be initialized to 0 (even if it is ignored by E2E Library).
Description:	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 11. For each transmitted Data, there is an instance of this typedef.		
Available via:	E2E.h		

E2E_P11ProtectStateType

Name:	E2E_P11ProtectStateType		
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 11.		
Available via:	E2E.h		

E2E_P11CheckStateType

Name:	E2E_P11CheckStateType
Type:	Structure

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

E2E_P11CheckStatusType

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

6.1.8 E2E Profile 22 types

E2E_P22ConfigType

Name:	E2E_P22ConfigType		
Type:	Structure		
Element:	uint16	DataLength	Length of Data, in bits. The value shall be a multiple of 8.
	uint8[16]	DataIDList	An array of appropriately chosen Data IDs for protection against masquerading.

	uint8	MaxDeltaCounter	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.
	uint16	Offset	Offset of the E2E header in the Data[] array in bits. It shall be: 0 ≤ Offset ≤ MaxDataLength-(2*8).
Description:	Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 22. The position of the counter and CRC is not configurable in profile 22.		
Available via:	E2E.h		

E2E_P22ProtectStateType

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

E2E_P22CheckStateType

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

E2E_P22CheckStatusType

Name:	E2E_P22CheckStatusType		
Type:	--		
Range:	E2E_P22STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.

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

6.1.9 E2E state machine types

E2E_PCheckStatusType

Name:	E2E_PCheckStatusType		
Type:	--		
Range:	E2E_P_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check).
	E2E_P_REPEATED	0x01	Data has a repeated counter.
	E2E_P_WRONGSEQUENCE	0x02	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
	E2E_P_ERROR	0x03	Error not related to counters occurred (e.g. wrong CRC, wrong length, wrong Data ID) or the return of the check function was not OK.

	E2E_P_NOTAVAILABLE	0x04	No value has been received yet (e.g. during initialization). This is used as the initialization value for the buffer, it is not returned by any E2E profile.
	E2E_P_NONEWDATA	0x05	No new data is available.
	reserved	0x07, 0x0F	Reserved for runtime errors (shall not be used for any status in future).
Description:	Profile-independent status of the reception on one single Data in one cycle.		
Available via:	E2E.h		

E2E_SMConfigType

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

6.2 Macro Constants

6.2.1 Error Flags by E2E Library

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

Type or error or status	How do caller of E2E shall handle it	Related code	Value [hex]
At least one pointer parameter is a NULL pointer	Development error or Integration error	E2E_E_INPUTERR_NULL	0x13
At least one input parameter is erroneous, e.g. out of range	Development error or Integration error	E2E_E_INPUTERR_WRONG	0x17
An internal library error has occurred (e.g. error detected by program flow monitoring, violated invariant or post condition)	Development error or Integration error	E2E_E_INTERR	0x19
Function completed successfully	NONE.	E2E_E_OK	0x00
Function executed in wrong state	Development error or integration error	E2E_E_WRONGSTATE	0x1A

6.2.2 Error Flags by E2E Protection Wrapper functions on byte 3 of the return value

Type or error or status	How should the caller of E2E Wrapper handle it	Related code	Value [hex]
OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.	Production	E2EPW_STATUS_OK	0x0
Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.	Production	E2EPW_STATUS_NONEW DATA	0x1
Error: The data has been received according to communication medium, but the CRC or Data or part of Data is incorrect/corrupted. This may be caused by corruption, insertion or by addressing faults.	Production	E2EPW_STATUS_WRONG CRC	0x2
NOT VALID: The new data has been received after detection of an unexpected 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	Production	E2EPW_STATUS_SYNC	0x3
Error: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.	Production	E2EPW_STATUS_INITIAL	0x4
Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.	Production	E2EPW_STATUS_REPEA TED	0x8
OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter ≤ MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.	Production	E2EPW_STATUS_OKSOM ELOST	0x20
Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.	Production	E2EPW_STATUS_WRONG SEQUENCE	0x40

6.3 Functions

6.3.1 E2E Library – E2E Profile 1 routines

6.3.1.1 E2E_P01Protect

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 (inout):	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID.	
Available via:	E2E.h	

6.3.1.2 E2E_P01ProtectInit

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

6.3.1.3 E2E_P01Check

Service name:	E2E_P01Check	
Syntax:	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.
Available via:	E2E.h

6.3.1.4 E2E_P01CheckInit

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

6.3.1.5 E2E_P01MapStatusToSM

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

6.3.2 E2E Library – E2E Profile 2 routines

6.3.2.1 E2E_P02Protect

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

6.3.2.2 E2E_P02ProtectInit

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

6.3.2.3 E2E_P02Check

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

6.3.2.4 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):	None	
Parameters (out):	StatePtr	Pointer to port/data communication state.
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL - null pointer passed E2E_E_OK
Description:	Initializes the check state	
Available via:	E2E.h	

6.3.2.5 E2E_P02MapStatusToSM

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

6.3.3 E2E Library – E2E Profile 4 routines

6.3.3.1 E2E_P04Protect

Service name:	E2E_P04Protect	
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.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description:	Protects the array/buffer to be transmitted using the E2E profile 4. This includes checksum calculation, handling of counter and Data ID.	
Available via:	E2E.h	

6.3.3.2 E2E_P04ProtectInit

Service name:	E2E_P04ProtectInit	
Syntax:	Std_ReturnType E2E_P04ProtectInit(E2E_P04ProtectStateType* StatePtr)	
Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	StatePtr	Pointer to port/data communication state.
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	
Available via:	E2E.h	

6.3.3.3 E2E_P04Check

Service name:	E2E_P04Check	
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.	
Available via:	E2E.h	

6.3.3.4 E2E_P04CheckInit

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

6.3.3.5 E2E_P04MapStatusToSM

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

6.3.4 E2E Library – E2E Profile 5 routines

6.3.4.1 E2E_P05Protect

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

6.3.4.2 E2E_P05ProtectInit

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

6.3.4.3 E2E_P05Check

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

6.3.4.4 E2E_P05CheckInit

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

6.3.4.5 E2E_P05MapStatusToSM

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

6.3.5 E2E Library – E2E Profile 6 routines

6.3.5.1 E2E_P06Protect

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

6.3.5.2 E2E_P06ProtectInit

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

6.3.5.3 E2E_P06Check

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

6.3.5.4 E2E_P06CheckInit

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

6.3.5.5 E2E_P06MapStatusToSM

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

6.3.6 E2E Library – E2E Profile 7 routines

6.3.6.1 E2E_P07Protect

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.	
Available via:	E2E.h	

6.3.6.2 E2E_P07ProtectInit

Service name:	E2E_P07ProtectInit	
Syntax:	Std_ReturnType E2E_P07ProtectInit(E2E_P07ProtectStateType* StatePtr)	
Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	StatePtr	Pointer to port/data communication state.
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	
Available via:	E2E.h	

6.3.6.3 E2E_P07Check

Service name:	E2E_P07Check	
Syntax:	Std_ReturnType E2E_P07Check(const E2E_P07ConfigType* ConfigPtr, E2E_P07CheckStateType* StatePtr, const uint8* DataPtr, uint32 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 7. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via:	E2E.h	

6.3.6.4 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	
Available via:	E2E.h	

6.3.6.5 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.	
Available via:	E2E.h	

6.3.7 E2E Library – E2E Profile 11 routines

6.3.7.1 E2E_P11Protect

Service name:	E2E_P11Protect	
Syntax:	Std_ReturnType E2E_P11Protect(const E2E_P11ConfigType* ConfigPtr, E2E_P11ProtectStateType StatePtr, uint8 DataPtr, uint16 Length)	
Service ID[hex]:	0x3b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout):	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_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 11. This includes checksum calculation, handling of counter.	
Available via:	E2E.h	

6.3.7.2 E2E_P11ProtectInit

Service name:	E2E_P11ProtectInit	
Syntax:	Std_ReturnType E2E_P11ProtectInit(E2E_P11ProtectStateType* StatePtr)	
Service ID[hex]:	0x3c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	StatePtr	Pointer to port/data communication state.
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	
Available via:	E2E.h	

6.3.7.3 E2E_P11Check

Service name:	E2E_P11Check	
Syntax:	Std_ReturnType 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:	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 11. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via:	E2E.h	

6.3.7.4 E2E_P11CheckInit

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

6.3.7.5 E2E_P11MapStatusToSM

Service name:	E2E_P11MapStatusToSM	
Syntax:	E2E_PCheckStatusType E2E_P11MapStatusToSM(Std_ReturnType CheckReturn, E2E_P11CheckStatusType Status)	
Service ID[hex]:	0x3a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CheckReturn	Return value of the E2E_P11Check function
	Status	Status determined by E2E_P11Check 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 11 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 11 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via:	E2E.h	

6.3.8 E2E Library – E2E Profile 22 routines

6.3.8.1 E2E_P22Protect

Service name:	E2E_P22Protect	
Syntax:	Std_ReturnType 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:	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 22. This includes checksum calculation, handling of counter.	
Available via:	E2E.h	

6.3.8.2 E2E_P22ProtectInit

Service name:	E2E_P22ProtectInit
Syntax:	Std_ReturnType 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:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the protection state.	
Available via:	E2E.h	

6.3.8.3 E2E_P22Check

Service name:	E2E_P22Check	
Syntax:	Std_ReturnType 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:	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 22. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via:	E2E.h	

6.3.8.4 E2E_P22CheckInit

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

6.3.8.5 E2E_P22MapStatusToSM

Service name:	E2E_P22MapStatusToSM	
Syntax:	E2E_PCheckStatusType 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):	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
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.	
Available via:	E2E.h	

6.3.9 E2E Library – E2E State Machine routines

6.3.9.1 E2E_SMCheck

Service name:	E2E_SMCheck	
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 (inout):	StatePtr	Pointer to port/data communication state.
Parameters (out):	None	
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_INTERR E2E_E_OK E2E_E_WRONGSTATE For definitions for return values, see SWS_E2E_00047.
Description:	Checks the communication channel. It determines if the data can be used for safety-related application, based on history of checks performed by a corresponding E2E_P0XCheck() function.	
Available via:	E2E.h	

6.3.9.2 E2E_SMCheckInit

Service name:	E2E_SMCheckInit	
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 (inout):	None	
Parameters (out):	StatePtr	Pointer to port/data communication state.
Return value:	Std_ReturnType	E2E_E_INPUTERR_NULL – null pointer passed E2E_E_OK
Description:	Initializes the state machine.	
Available via:	E2E.h	

6.3.10 E2E Protection Wrapper

6.3.10.1 Single channel wrapper routines and init routines

E2EPW_Write_<p>_<o>

Service name:	E2EPW_Write_<p>_<o>	
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:	uint32	The byte 0 (lowest byte) is the status of Rte_Write function: RTE_E_COM_STOPPED - the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only) RTE_E_SEG_FAULT - a segmentation violation is detected in the handed over parameters to the RTE API. No transmission is executed RTE_E_OK - data passed to communication service successfully The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or post condition) 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 post condition) E2E_E_OK - Function E2E_PXXProtect completed successfully The byte 3 is a placeholder for future use and takes the following values E2E_E_OK - default case
Description:	Initiates a safe explicit sender-receiver transmission of a safety-related data element with data semantic. It protects data with E2E Library function E2E_PXXProtect and then it calls the corresponding RTE_Write function.	
Available via:	E2E.h	

E2EPW_WriteInit_<p>_<o>

Service name:	E2EPW_WriteInit_<p>_<o>	
Syntax:	Std_ReturnType E2EPW_WriteInit_<p>_<o>(Rte_Instance <instance>)	
Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	Status of runtime checks: E2E_E_INTERR – An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or post condition) E2E_E_OK – Function completed successfully
Description:	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via:	E2E.h	

E2EPW_Read_<p>_<o>

Service name:	E2EPW_Read_<p>_<o>	
Syntax:	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 post condition) 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 post condition) 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 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.
Description:	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	
Available via:	E2E.h	

E2EPW_ReadInit_<p>_<o>

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

Reentrancy:	Non Reentrant	
Parameters (in):	<instance>	SW-C instance. This parameter is not used (it is ignored).
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	Status of runtime checks: E2E_E_INTERR - An internal error has occurred in the function (e.g. error detected by program flow monitoring, violated invariant or post condition) E2E_E_OK - Function completed successfully
Description:	The function reinitializes the corresponding data structure after a detected error or at startup.	
Available via:	E2E.h	

6.3.10.2 Redundant wrapper routines

E2EPW_Write1_<p>_<o>

Service name:	E2EPW_Write1_<p>_<o>	
Syntax:	uint32 E2EPW_Write1_<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:	uint32	The byte 0 (lowest byte) is equal to E2E_E_OK (because Rte_Write is not invoked) The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Write is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Write is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or post condition) 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 post condition) E2E_E_OK - Function E2E_PXXProtect completed successfully The byte 3 is a placeholder for future use and takes the following values: E2E_E_OK - default case
Description:	It protects data with E2E Library function E2E_PXXProtect. It does not call the corresponding RTE_Write function.	
Available via:	E2E.h	

E2EPW_Write2_<p>_<o>

Service name:	E2EPW_Write2_<p>_<o>

Syntax:	uint32 E2EPW_Write2_<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.
	<data>	Data element to be protected and sent. The parameter is IN, because this function compares the calculated protection fields from E2EPW_Write1 with independently calculated fields from invoking E2E_PXXProtect. Nothing is changed in <data> in case of success. The name and data type are the same as in the corresponding Rte_Write function.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint32	The byte 0 (lowest byte) is the status of Rte_Write function: RTE_E_COM_STOPPED – the RTE could not perform the operation because the COM service is currently not available (inter ECU communication only) RTE_E_SEG_FAULT – a segmentation violation is detected in the handed over parameters to the RTE API. No transmission is executed RTE_E_OK – data passed to communication service successfully The byte 1 is the status of runtime Protects done within E2E Protection Wrapper function: E2E_E_INPUTERR_NULL – At least one pointer parameter of E2EPW_Write is a NULL pointer E2E_E_INPUTERR_WRONG – At least one input parameter of E2EPW_Write is erroneous, e.g. out of range E2E_E_INTERR – An internal error has occurred in E2EPW_Write (e.g. error detected by program flow monitoring, violated invariant or post condition) 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 post condition) 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.
Available via:	E2E.h

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

E2EPW_Read1_<p>_<o>

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_NEVER_RECEIVED - No data received since system start or partition restart RTE_E_UNCONNECTED - Indicates that the receiver port is not connected.
		RTE_E_OK - data read successfully The byte 1 is the status of runtime checks done within E2E Protection Wrapper function: E2E_E_INPUTERR_NULL - At least one pointer parameter of E2EPW_Read is a NULL pointer E2E_E_INPUTERR_WRONG - At least one input parameter of E2EPW_Read is erroneous, e.g. out of range E2E_E_INTERR - An internal error has occurred in E2EPW_Read (e.g. error detected by program flow monitoring, violated invariant or post condition)

		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 post condition) 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 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.
Description:	Performs a safe explicit read on a sender-receiver safety-related communication data element with data semantics. The function calls the corresponding function RTE_Read, and then checks received data with E2E_PXXCheck.	
Available via:	E2E.h	

E2EPW_Read2_<p>_<o>

Service name:	E2EPW_Read2_<p>_<o>	
Syntax:	uint32 E2EPW_Read2_<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.
	<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 post condition) 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 post condition) 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 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.
Description:	The function re-checks the data received with corresponding function Read1 by means of execution of E2E_PXXCheck.	
Available via:	E2E.h	

E2EPW_ReadInit1_<p>_<o>

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

E2EPW_ReadInit2_<p>_<o>

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

6.3.11 COM E2E Callouts

6.3.11.1 IPDU_E2EProtect_<IPDU ID> ()

Function Name	IPDU_E2EProtect_<IPDU ID>
Syntax:	boolean IPDU_E2EProtect_<IPDU ID>(PduIdType PduId, PduInfoType* PduInfoPtr)
Service ID	NONE.
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (In)	PduId: I-PDU ID
Parameters (Inout)	PduInfoPtr: I-PDU 데이터의 포인터
Parameters (Out)	None
Return Value	boolean: true – Data protection is completed through the E2E_PXXProtect function false – An error occurred while running E2E_PXXProtect
Description	Protection is performed on the array / buffer to be transmitted using E2E through ComIPduCallout.
Preconditions	None
Configuration Dependency	None

6.3.11.2 IPDU_E2ECheck_<IPDU ID>()

Function Name	IPDU_E2ECheck_<IPDU ID>
Syntax:	boolean IPDU_E2ECheck_<IPDU ID>(PduIdType PduId, PduInfoType* PduInfoPtr)
Service ID	NONE.
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (In)	PduId: I-PDU ID PduInfoPtr: Pointer to I-PDU data
Parameters (Inout)	None
Parameters (Out)	None
Return Value	boolean
Description	The data received using E2E through ComIPduCallout is checked.
Preconditions	None
Configuration Dependency	None

7. Generator

7.1 Generator Option

7.1.1 E2E

None

7.1.2 Usage of E2E Library (Rte)

Option	Description
Bend	Add this option when the CPU's Byte Order is Big Endian (High Byte First).

7.2 Generator Error Message

7.2.1 E2E

7.2.1.1 Error Messages

None

7.2.1.2 Warning Messages

None

7.2.1.3 Information Messages

None

8. Appendix

None