Elektrobit

EB tresos®Safety E2E Wrapper safety manual

EB tresos ®Safety Wrapper

Date: 2019-11-22, ID: EBASCE2ESE-16, Document version: 3.3, Status: RELEASED

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Confidential information.

Table of Contents

1. Document History

Version	Date	Author	State	Description
2.5	2016-12-20	Elektrobit Automotive GmbH	DRAFT	apply new template
2.6	2017-03-30	Elektrobit Automotive GmbH	DRAFT	Remove profile 1 and 2, update due to review findings
2.7	2017-08-21	Elektrobit Automotive GmbH	DRAFT	Remove E2EPW configuration details update due to review findings
3.0	2017-10-02	Elektrobit Automotive GmbH	PROPOSED	Update due to review findings
3.0	2017-10-02	Elektrobit Automotive GmbH	RELEASED	Set to released, ASCE2ESE-685
3.1	2017-12-12	Elektrobit Automotive GmbH	PROPOSED	Update of the safety check list
3.1	2017-12-12	Elektrobit Automotive GmbH	RELEASED	Set to released, ASCE2ESE-694
3.2	2018-02-13	Elektrobit Automotive GmbH	RELEASED	update verification of delivery content via SHA-1 hashes
3.3	2019-04-12	Elektrobit Automotive GmbH	DRAFT	Set status to draft
3.3	2019-11-22	Elektrobit Automotive GmbH	RELEASED	Set status to released, no changes

Table 1.1. Document history

2. Document information

2.1. Objective

The objective of this document is to provide you with all the information necessary to ensure that the *EB tresos Safety E2E Wrapper* (*E2EPM*) is used in a safe way in your project.

The goal of the safety manual is to provide the following information:

- Definition of the *EB tresos Safety E2E Wrapper*
- Information and requirements for the integration of the *EB tresos Safety E2E Wrapper* into an item
- Information and requirements for the use of the *EB tresos Safety E2E Wrapper* in an item
- Requirements from the *EB tresos Safety E2E Wrapper* on the item
- Information about the customer interfaces of *EB tresos Safety E2E Wrapper* supporting processes

2.2. Scope and audience

This manual describes the usage of *EB tresos Safety E2E Wrapper* in system applications which have safety allocations up to ASIL-D. It is valid for all projects and organizations which use *EB tresos Safety E2E Wrapper* in a safety-related environment. The *EB tresos Safety E2E Wrapper* is intended to be used in AUTOSAR ECU projects.

The intended audience of this document is:

Professionals in embedded automotive systems with the appropriate qualification in the area of functional safety, communication networks, and AUTOSAR.

2.3. Quality and Safety Statement

Information about the quality level of an *EB tresos Safety E2E Wrapper* release is provided in the Quality Statement. If such a statement is not available the software shall be considered as Prototype and must not be used in mass production.

2.4. Motivation

This manual provides the information on how to correctly use the *EB tresos Safety E2E Wrapper* in projects for safety-related environments. If the *EB tresos Safety E2E Wrapper* is used differently, the *EB tresos Safety E2E Wrapper* code might not comply with the assumed safety requirements, which are defined in Section 3.2.3, “Assumed safety requirements of the EB tresos Safety E2E Wrapper”.

2.5. Structure

- Chapter 2, “Document information“: (this chapter) provides a brief introduction of this document and its structure.
- Chapter 3, “About the EB tresos Safety E2E Wrapper“: introduces E2EPM and the environment in which it can be used, describes safety requirements, assumptions, and limitations.
- Chapter 4, “Using the EB tresos Safety E2E Wrapper safely“: describes how to correctly use the E2EPM.
- Chapter 5, “Safety element out of context (SEooC) definition“: describes the Safety Element out of Context (SCooC) E2EPM.
- Chapter 6, “Configuration verification criteria“: describes the verification criteria of the E2EPM.
- Appendix A, “Safety Checklist“: describes all required activities that need to be performed for a safe usage of the E2EPM.
- Appendix B, “Safety Checker“: describes the usage of the *Safety Transformer Checker*.

3. About the EB tresos Safety E2E Wrapper

The *EB tresos Safety E2E Wrapper* facilitates the exchange of safety-related data between AUTOSAR SWCs by protecting the data exchange against the effects of faults along the communication path including random HW faults, and systematic software faults.

3.1. Architecture of the surrounding system

3.1.1. Technical Overview

The exchange of data between two SW-C components over an unprotected communication link (consisting of COM SW and COM HW at the sender and the receiver, and Physical COM Link) may be affected by faults (Figure 3.1, “Unprotected Communication”). Such faults in the communication link can affect the integrity of exchanged data which may lead to a violation of the safety goal if such data is safety-related. Possible faults are:

- random HW faults (e.g. corrupt registers of a FlexRay controller),
- transient faults (e.g. interference due to EMC on the physical link), and
- systematic faults within the communication software (e.g. bug in COM module).

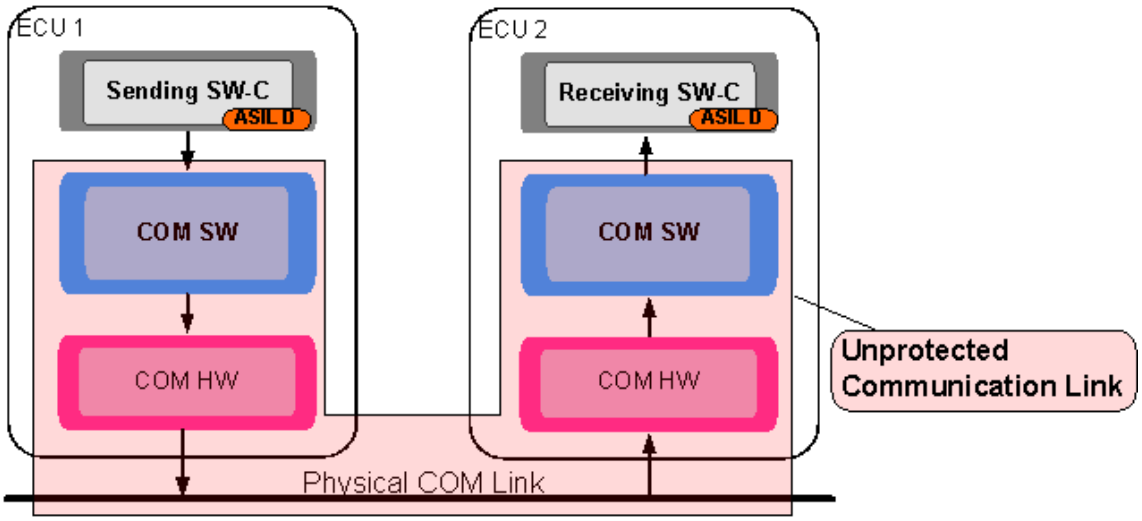

Figure 3.1. Unprotected Communication

According to ISO 26262, part 6, clause D.2.4 [ISO26262_1ST] the causes for faults or effects of faults such as those listed below can be considered for each sender or each receiver with respect to the exchange of information (i.e. such faults can cause interference between software elements):

Id	Failure mode name
1	repetition of information
2	loss of information
3	delay of information
4	insertion of information
5	masquerade or incorrect addressing of information
6	incorrect sequence of information
7	corruption of information
8	asymmetric information sent from a sender to multiple receivers
9	information from a sender received by only a subset of the receivers
10	blocking access to a communication channel

Table 3.1. Causes for communication faults or effects of faults.

3.1.2. Failure Model of E2E Communication

3.1.2.1. Object Definitions

Term	Description	Synonyms
Data source	In the E2EPM, a *data source* is defined as an object which produces data.	Sender
Data sink	In the E2EPM, a *data sink* is defined as an object which consumes data.	Receiver
Communication link	In the E2EPM, a *communication link* is defined as a link between a data source and a data sink for the purpose of forwarding messages from the data source to the data sink. Note 1: Hereinafter, we always consider communication links with safety related data sinks, because an E2E protection is usually used in the context of safety related applications. Note 2: In ASR, the hierarchical level of a data source and data sink of a communication link can either be at the level of Complex Data Elements, or at the level of COM IPdus. In the case of the level	communication channel

Term	Description	Synonyms
	of Complex Data Elements, the data source and data sink are defined by the triple <SW-C Instance, Port Instance, Data Element Instance>.	
Fault-free communication link	In the E2EPM, a *fault-free communication link* is defined as a communication link in the absence of faults.	
Message	In the E2EPM, a *message* is defined as data to be transmitted over a communication link.	
Safety related message	In the E2EPM, a *safety related message* is defined as a message that contains safety related data.	
Protected message	In the E2EPM, a *protected message* is defined as a message containing user data and protection information (e.g. CRC, sequence counter, etc.).	
Valid message	In the E2EPM, a *valid message* is defined as a protected message where the CRC fits to the user data of the message.	
Authenticated data source	In the E2EPM, an *authenticated data source* is defined as a data source which is uniquely identifiable by a data sink.	
Authentic message	In the E2EPM, an *authentic message* is defined as a message where the user data is identified as originated from the authenticated data source.	
User	In the E2EPM, a *user* is defined as the SWC which calls the E2EPM.	
Integrator	In the E2EPM, an *integrator* is defined as the person who integrates or configures the E2EPM.	
Communication error	In the E2EPM, a *communication error* is defined as occurrence of any E2E Communication Failure mode specified in Section 3.1.2.3, “E2E Communication Link Failure Modes”	
Message sequence indicator	In the E2EPM, a *message sequence indicator* is defined as a number representing the order of a message within a sequence of messages. For any two consecutively sent messages, n represents the first message and n+1 the second message.	
MaxSequenceCounter	In the E2EPM, the parameter *MaxSequenceCounter* is defined as the maximum allowed value of the *Message sequence indicator*. Note: An infinite maximum number is not possible due to resource restrictions.	
DeltaCounter	In the E2EPM, the parameter *DeltaCounter* is defined as the difference of the *Message sequence indicator* of a received valid and au-	

Term	Description	Synonyms
	thentic message with respect to the last received valid and authentic message.	
Message loss window	In the E2EPM, a *message loss window* defines a *window size* where received valid and authentic messages are accepted if and only if the *DeltaCounter* is greater than zero and smaller than or equal to the *window size* plus one.	
MaxDeltaCounterInit	In the E2EPM, the parameter *MaxDeltaCounterInit* is defined as the initial *window size* of the *Message loss window*.	
MaxDeltaCounter	In the E2EPM, the parameter *MaxDeltaCounter* is defined as the actual window size of the *Message loss window* for a specific message.	
Correct message	In the E2EPM, a *correct message* is defined as a valid message (if authentication is not required - depends on the context) or a valid and authentic message (if authentication is required - depends on the context) which is accepted by the *Message loss window*.	
Counter Continuity Check	In the E2EPM, a *Counter Continuity Check* is defined as an internal E2EPM stabilization process at the receiver, during which the deterministic detection of a correct message cannot be guaranteed.	
MaxNoNewOrRepeatedData	In the E2EPM, the parameter *MaxNoNewOrRepeatedData* is defined as the maximum allowed number of continuously received messages with the failure modes *message loss* or *unintended message repetition* after which a deterministic detection of a correct message can be guaranteed.	
Max message loss window	In the E2EPM, the *max message loss window* is defined as the maximum allowed window size of the *message loss window* which equals *MaxDeltaCounterInit*+ *MaxNoNewOrRepeatedData*-1 with the constraint that (*MaxDeltaCounterInit*+ *MaxNoNewOrRepeatedData*<= *MaxSequenceCounter*). Note: If the *message loss window* exceeds the *max message loss window*, then a deterministic detection of a correct message cannot be guaranteed anymore.	
SyncCounterInit	In the E2EPM, the parameter *SyncCounterInit* is defined as the number of Data required for validating the consistency of the counter that must be received with a valid counter after the detection of an unexpected behavior.	

3.1.2.2. Process Terms

Term	Description	Synonyms
to transmit	In the E2EPM, *to transmit* is defined as the process of putting a message into the communication link.	send
to receive	In the E2EPM, *to receive* is defined as the process of getting a message out of the communication link.	deliver
to protect	In the E2EPM, *to protect* is defined as the process of adding protection information (e.g. CRC, double inverse data) to a message.	
to indicate	In the E2EPM, *to indicate* is defined as the process of returning information (e.g. error information) back to the caller of the E2EPM.	
to (immediately) activate	In the E2EPM, *to activate* is defined as the process of starting a specific process state such that it is *active* afterwards. *Immediately* means, if a state change of a process occurred due to a message reception, the new state is assumed for this message (e.g. for message indication).	
to (immediately) de-activate	In the E2EPM, *to de-activate* is defined to end an *active* process state such that it is *de-activated* afterwards. *Immediately* means, if a state change of a process occurred due to a message reception, the new state is assumed for this message (e.g. for message indication).	
to (immediately) re-activate	In the E2EPM, *to re-activate* is defined as the process of first de-activating and immediately afterwards activating the specific process again such that it is *active* afterwards (e.g. for the re-initialization of parameters). *Immediately* means, if a state change of a process occurred due to a message reception, the new state is assumed for this message (e.g. for message indication).	

3.1.2.3. E2E Communication Link Failure Modes

In the following, the failure modes with respect to an E2E communication link are defined with a mapping to the ISO failure mode id of Table 3.1, “Causes for communication faults or effects of faults.”.

Failure mode	ISO failure mode id	Description
Unintended message repetition	1	In the E2EPM, *unintended message repetition* is defined as receiving a message m more than once on the communication link where m has been transmitted once.

Failure mode	ISO failure mode id	Description
Message loss	2	In the E2EPM, *message loss* is defined as never receiving message m on the communication link where m has been transmitted.
Insertion of messages	4	In the E2EPM, *insertion of messages* is defined as receiving an unprotected message m on the communication link where m has not been transmitted. Note: Insertion of messages cannot be clearly identified at the receiver. Since protection information (i.e. CRC) is assumed to be missing, the interpretation of message m as it would contain the protection information leads to the detection of a *message corruption* instead.
Re-sequencing	6	In the E2EPM, *Re-sequencing* is defined as receiving message m2 before message m1 on the communication link where message m1 has been transmitted before m2.
Message corruption	7	In the E2EPM, *message corruption* is defined as receiving m' instead of m and m' is not equal to m on the communication link where m has been transmitted.
Delayed reception	3	In the E2EPM *delayed reception* is defined as receiving a message m at tR, where tR > tE > tS, tE is the latest expected reception time and tS is the time m has been transmitted.
Addressing fault	5	In the E2EPM, *addressing fault* is defined as receiving a message m on a communication link B instead of the communication link A where m has been transmitted.
Masquerading	5	In the E2EPM, *masquerading* is defined as receiving a non-authentic message m which appears authentic on the communication link where m has been transmitted or where m has been expected to be transmitted in case the data source is identified by the communication link. Note: The latter case results from an addressing fault.

Note:

For End-To-End communication the failure mode ids 8 and 9 of Table 3.1, “Causes for communication faults or effects of faults.” are not relevant as they are not related to a point-to-point communication scenario. Failure mode id 10 of Table 3.1, “Causes for communication faults or effects of faults.” ("blocking access to a communication channel") results in the failure mode message delay (i.e. other messages cannot be received) and, therefore, no additional detection mechanism is required.

3.1.3. E2E Protection in Autosar

The detection of these failure modes at the receiving SW-C requires the integration of safety mechanisms that includes the transmission of related protection information to the safety-related user data by the sending SW-C and evaluation of this protection information data by the receiving SW-C (Figure 3.2, “Protected Communication”). Therefore AUTOSAR has specified a standard AUTOSAR library [ASRSWSE2E] which implements safety mechanisms for protected communication according to specific E2E Profiles.

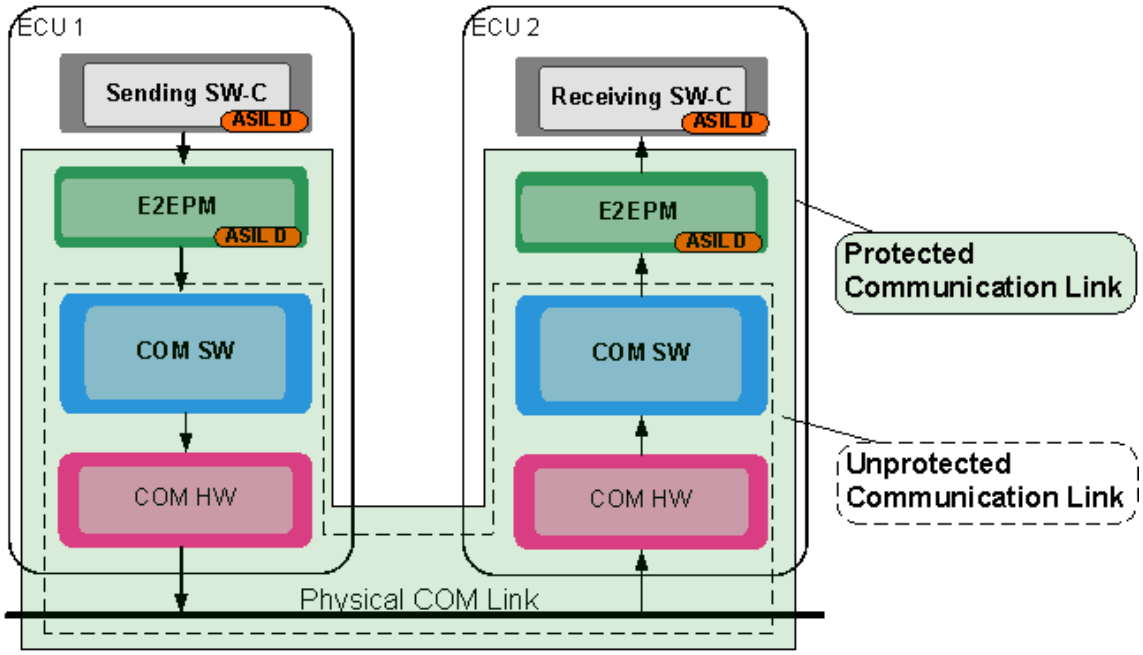

Figure 3.2. Protected Communication

A safety mechanism in the E2EPM is a functionality with the purpose to detect communication faults. This includes functionality at the sender that usually adds protection information to the payload data, as well as functionality at the receiver side that processes the received information in order to detect a communication fault.

This E2EPM implements those safety mechanisms and is located between the Sending SW-C and the COM SW (RTE and ASR Communication Stack) respectively between the Receiving SW-C and the COM SW.

3.2. Description of the EB tresos Safety E2E Wrapper

3.2.1. Identification of the EB tresos Safety E2E Protection

The *EB tresos Safety E2E Wrapper* is composed of the following products

- *EB tresos Safety E2E Wrapper*
- EB tresos Safety E2E Profiles (e.g. EB tresos Safety E2E Profile 2)

and a *E2EPW Checker tool* and the following user documentation:

- E2EPW user documentation, E2E_E2EPW_documentation
- User documentation for the used E2E profiles
- Safety Manual, this document.

3.2.2. Functional requirements of the EB tresos Safety E2E Wrapper

The *EB tresos Safety E2E Wrapper* is developed as a Safety Element out of Context (SEooC) according to INTERNATIONAL STANDARD ISO 26262 [ISO26262_1ST]. Refer to Section 3.2.3, “Assumed safety requirements of the EB tresos Safety E2E Wrapper”.

3.2.3. Assumed safety requirements of the EB tresos Safety E2E Wrapper

The E2EPM is a Safety Element out of Context (SEooC) on software level and implements the following assumed safety requirements. The correct implementation of these requirements has been verified.

[EB_E2ESE060002]

The user of the E2EPM has to validate that the assumed E2EPM requirements comply with the user's safety concept.

Note:

Note: non-safety related requirements of the E2EPM are not listed here.

3.2.3.1. Top level requirements for the E2E Protection Module

3.2.3.1.1. EB_E2ESE000070: Freedom from Interference of the data element exchange (S)

Id:	**EB_E2ESE000070**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Freedom from Interference of the data element exchange
Priority:	HIGH
Description:	The E2EPM shall provide Freedom from Interference of the data element exchange between two AUTOSAR SWCs.
Rationale:	End-to-end communication protection is state-of-art in safety-related automotive systems to ensure freedom from interference of the SW-Cs from the communication link between them. Detection of communication link failure modes are required to ensure freedom from interference.
Use Case:	End-to-End communication protection for communication on an unsafe communication link.
Dependencies:	None
Supporting Material:	
Verification criteria:	All relevant communication link failure modes are detected by E2EPM.
Comment:	

3.2.3.1.2. EB_E2ESE000076: Freedom from Interference regarding memory (S)

Id:	**EB_E2ESE000076**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Freedom from Interference regarding memory
Priority:	HIGH
Description:	The E2EPM shall provide Freedom from Interference of the E2EPM memory
Rationale:	Protection of E2EPM internal variables and configuration against corruption resulting from faults in another software.
Use Case:	Coexistence of safety related and non safety related components or safety related components with different ASIL.
Dependencies:	None
Supporting Material:	
Verification criteria:	A corruption of E2EPM internal variables and configuration is detected by E2EPM.
Comment:	

3.2.3.2. Claims for the E2E Protection Module

3.2.3.2.1. EB_E2ESE000071: Protection of data element exchange (S)

Id:	**EB_E2ESE000071**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Protection of data element exchange
Priority:	HIGH
Description:	The E2EPM shall provide protection of data element exchange between two AUTOSAR SWCs.
Rationale:	In order to detect communication link failures the data element has to be protected.
Use Case:	End-to-End communication protection for communication on an unsafe communication link.
Dependencies:	None
Supporting Material:	
Verification criteria:	The data element exchange has been protected by adding appropriate protection information.
Comment:	

3.2.3.2.2. EB_E2ESE000072: Communication failure detection (S)

Id:	**EB_E2ESE000072**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Communication failure detection
Priority:	HIGH
Description:	The E2EPM shall provide communication failure detection of protected data element exchange between two AUTOSAR SWCs.
Rationale:	Detection of communication link failures.
Use Case:	End-to-End communication protection for communication on an unsafe communication link.
Dependencies:	None
Supporting Material:	
Verification criteria:	All relevant communication link failure modes are detected by the E2EPM.
Comment:	

3.2.3.2.3. EB_E2ESE000073: Configurable tolerance level (S)

Id:	**EB_E2ESE000073**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Configurable tolerance level
Priority:	HIGH
Description:	The E2EPM shall provide a configurable tolerance level regarding message sequence discontinuities and automatic resynchronization for a higher availability of the data element exchange.
Rationale:	Robust communication between two SWCs
Use Case:	Communication between applications which tolerate lost messages.
Dependencies:	None
Supporting Material:	
Verification criteria:	Communication is not interrupted due to message sequence discontinuities.
Comment:	

3.2.3.2.4. EB_E2ESE000074: Source specific error indication (S)

Id:	**EB_E2ESE000074**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Source specific error indication
Priority:	HIGH
Description:	The E2EPM shall provide source specific error indication to related AUTOSAR SWC.
Rationale:	Detected errors cannot be handled by the E2EPM and thus must be indicated to the E2EPM User.
Use Case:	The application performs individual error handling of the errors indicated by the E2EPM.
Dependencies:	None
Supporting Material:	
Verification criteria:	Source specific errors are indicated to the E2EPM User via a function return value.
Comment:	

3.2.3.2.5. EB_E2ESE000075: Self-protection mechanism (S)

Id:	**EB_E2ESE000075**
Version:	1
Source:	EB
Status:	approved
Safety class:	ASIL-D
Short Description:	Self-protection mechanism
Priority:	HIGH
Description:	The E2EPM shall provide self-protection mechanism for environments which cannot guarantee freedom from interference regarding memory.
Rationale:	Protection of E2EPM internal variables and configuration against corruption resulting from faults in another software.
Use Case:	Coexistence of safety related and non safety related components or safety related components with different ASIL.
Dependencies:	None
Supporting Material:	
Verification criteria:	A corruption of E2EPM internal variables and configuration is detected by E2EPM.
Comment:	

3.2.4. Safety mechanism used by the EB tresos Safety E2E Wrapper

3.2.4.1. E2EPM Safety Mechanisms

3.2.4.1.1. E2EPM Communication Protection Mechanism

A communication protection mechanism in the E2EPM is a functionality with the purpose that a receiver detects communication faults. This includes functionality at the sender that adds Protection Information (PI) to the data for transmission, as well as functionality at the receiver that evaluates the the received PI in order to detect and indicate communication faults.

[EB_E2ESE003000]

The data and control flow for the E2EPM Communication Protection Mechanism is depicted in Figure 3.3, “Data flow and control flow of the communication protection mechanism”.

Legend: The control flow is depicted with solid arrows in red color and the data flow with dotted arrows in blue color. Oval shapes in red represent control flow actions. Solid boxes in blue represent persistent data. Dotted boxes in blue represent intermediate data.

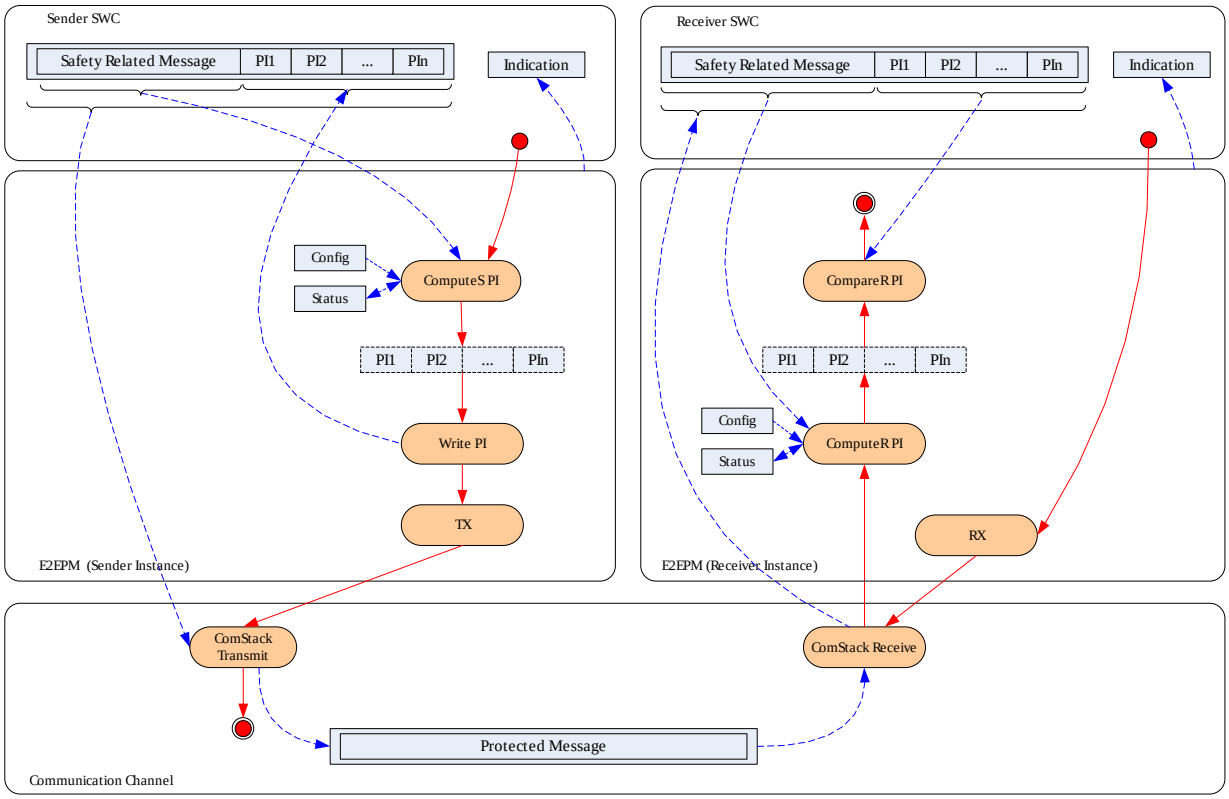

Figure 3.3. Data flow and control flow of the communication protection mechanism

[EB_E2ESE003010]

Components and their data structures:

- Sender SW-C
  - Safety Related Message: contains safety related data that should be transmitted over the communication channel
  - (PI1, ..., PIn): Protection Information (PI) (e.g. CRC, Sequence/Alive Counter, unique data ID)
  - Indication: indication of an application error (e.g range check) as well as internal errors of the E2EPM and external errors
- E2EPM (Sender Instance)
  - Config: data ID
  - Status: last sent sequence counter
  - PI: locally computed PI
- Communication Channel
  - Protected Message: contains the safety related Message with added PI

- E2EPM (Receiver Instance)
  - Config: data ID, Rx properties (e.g. MaxDeltaCounterInit, MaxNoNewOrRepeatedData, SyncCounterInit, etc.)
  - Status: last successfully received sequence counter, additional information for error detection (e.g. DeltaCounter, number of repeated data counter, sync counter, etc.)
  - PI: locally computed PI
- Receiver SW-C
  - Safety Related Message: contains the safety related data that was received over the communication channel
  - (PI1, ..., PIn): PI of the safety related data that was received (e.g. CRC, Sequence/Alive Counter, unique data ID)
  - Indication: indication of a communication error as well as internal errors of the E2EPM, or external errors

[EB_E2ESE003020]

Control Flow at the Sender SW-C:

- Sender SWC initiates the sending of the safety related payload.
- ComputeS PI: computes the PI based on
  - the safety related message,
  - the configuration data (e.g. data ID),
  - and the status information (e.g sequence counter)
- Write PI: writes the locally computed PI to the data container of the PI provided by the Sender-SWC.
- TX: the E2EPM passes the message to ComStack Transmit
- ComStack Transmit: Transmission of the serialized data
- Result Indication: Indication of an application error (e.g range check) as well as internal errors to the Sender-SWC.

[EB_E2ESE003030]

Control flow at the Receiver SWC:

- Receiver SW-C: cyclically initiates the reception of a safety related message
- RX: The SW-C initiates the E2EPM to pass the receive request to the ComStack receive function.
- ComStack Receive: copies the received protected message to the data container in the receiver SWC
- ComputeR PI: computes the expected PI based on
  - the safety related message,
  - the configuration data (e.g. data ID),
  - and the status information (e.g last received sequence counter)

- CompareR PI: compares the received PI with the expected protection information.
- Result Indication: Indicates the result (e.g. communication error) to the receiver SW-C

[EB_E2ESE003040]

Within the E2EPM the following safety mechanisms (PI and its corresponding mechanism) are used:

- CRC: the sender computes a CRC over counter, safety related data, and optionally over data ID. The receiver verifies the correctness.
- Counter: a counter that is incremented at every send request and its value is checked at receiver side for correct incrementation compared to the counter of the previously received protected message.
- Data ID: a unique number in order to identify the identity of each transmitted safety related message. The receiver verifies the correctness of the data ID of the received message. This data ID may be explicitly transmitted or indirectly by influencing the CRC.
- Timeout detection: the receiver periodically calls the E2EPM RX function and evaluates the return code. If the return code indicates that no new data have been received for x times in a row then a timeout of `x * invocation period` is detected. (The E2EPM provides means that support the timeout detection at the SW-C, thus the safety mechanisms timeout detection can also be implemented differently.)
- Range Check: the sender verifies that the data provided for transmission are within the configured ranges. The receiver verifies that the received data at the receiver SW-C are within the configured ranges.

Table 3.2, “Failure modes detection matrix” shows the required PI / safety mechanisms for the detection of a specific failure mode at the receiver. This mapping is based on Table 6 in [ASRSWSE2E]. The sufficiency of the safety mechanism to detect the specific failure modes has to be verified by the customer.

An 'X' specifies that the safety mechanisms detects or contributes to the detection of the related failure mode.

An (X) specifies that the safety mechanism contributes to the detection of the related failure mode.

An 'A' specifies that the failure mode is not detected directly by a detection mechanism of the E2EPM but can be detected at the receiver SW-C.

Failure Mode \ Safety Mechanism	Sequence counter	CRC	Data ID	Timeout detection	Range check
Unintended message repetition	X				
Message loss	X			A	
Insertion of message	X	(X)	X		

Failure Mode \ Safety Mechanism	Sequence counter	CRC	Data ID	Timeout detection	Range check
Resequencing	X				
Message corruption		X			X
Delayed reception				A	
Addressing faults	(X)	(X)	X		
Masquerading	(X)	(X)	X		

Table 3.2. Failure modes detection matrix

Depending on the used communication and network stack, appropriate subsets of theses mechanisms, so called profiles, can be grouped as for example listed in the following descriptions.

An E2E profile is a specific set of communication protection mechanisms for the detection of communication link failure modes. An E2E Profile is typically defined by an OEM or AUTOSAR.

[EB_E2ESE060001]

The user shall verify that the selected E2E profile is appropriate for the indented use case.
Note: Fulfilled by the userguide.

[EB_E2ESE060004]

The user has to ensure that the chosen E2E profile with its constrains (e.g. maximum permitted length of the protected data to ensure an appropriate error detection capability; transmission of one undetected erroneous data element in a sequence of data elements between sender and receiver will not lead to the violation of a safety goal of this system) fulfils the safety goals regarding communication protection.
Note: Fulfilled by the user guide.

3.2.4.1.2. Protection of pure Intra-ECU Communication

The aforementioned communication protection mechanisms are based on the existence of mapping information of the protected data elements to the PDU layout on the communication bus. Such a mapping information does not exist in the case of the protection of a pure Intra-ECU communication.

In general the E2E protection mechanism requires systematically serialized data. On this serialized data structure the protection mechanism is applied and finally the position where the protection information is stored has to be known. In the case of Intra-ECU communication the following constraints/pre-conditions and mechanisms are applied.

[EB_E2ESE003300]

Preconditions/Constraints for the algorithm:

- **IPC-PRE-1:** In case of intra-ECU and/or inter-Partition communication, the protection information within a complex data element shall be specified by the user, e.g. configuration of a prefix list for the Counter and CRC member.
- **IPC-PRE-2:** The protection mechanism used for intra-ECU and/or inter-Partition communication shall support the protection of a serialized data structure with different data length, i.e. a profile with a fixed data length cannot be used for pure Intra-ECU communication.

  In case of intra-ECU and/or inter-Partition communication, the generated Application Data Type (C-typedef) for the sender and receiver Data Elements shall be identical.
- **IPC-CON-1:** For this algorithm the range check cannot be applied.

[EB_E2ESE003301]

For the protection of pure intra-ECU communication for E2E Profiles that do not provide configurable positions of protection information, the following algorithm shall be applied:

- **IPC-1:** Copy the memory allocated by the complex data element byte-wise to a byte array with padding bytes set to 0.
- **IPC-2:** Swap the memory of the Crc struct member with the memory at the profile-specific position of the Crc (1 byte).
- **IPC-3:** Swap the memory of the Counter struct member with the memory at the profile-specific position of the Counter (1 byte).
- **IPC-4:** Apply the specified protection mechanism on the memory with the swapped Counter and Crc byte with data length set to the number of bytes of the memory allocated by the complex data element.

[EB_E2ESE003302]

For the protection of pure intra-ECU communication for E2E Profiles that do provide configurable positions of protection information (e.g. Profile 1), the following algorithm shall be applied:

- **IPC-E2EP01-1:** Copy the memory allocated by the complex data element byte-wise to a byte array with padding bytes either set to 0xFFU (E2E Profile 1 variants only) or 0 (E2E Profile VCC only).
- **IPC-E2EP01-2:** Apply the specified protection mechanism on the allocated memory with the following E2E Profile configuration data (setting of data length and offsets):
  - The data length shall be set to the number of bytes of the memory allocated by the complex data element.
  - The Counter offset shall be set to the bit-offset of the related Counter Data Element member within the complex data element type (is platform-dependent).
  - The CRC offset shall be set to the bit-offset of the related CRC Data Element member within the complex data element type (is platform-dependent).

- The DataIDNibble offset, if used, shall be set to the bit-offset of the related DataIDNibble Data Element member within the complex data element type (is platform-dependent).
- The DataIDNibble offset, if not used, shall be set to 0.

3.2.4.1.3. E2EPM Self Protection Mechanism

The E2EPM is embedded in an execution environment which can be harsh such that the environment interferes with the code or data of the E2EPM. To ensure freedom from interference of the E2EPM code and data memory from the execution environment, appropriate safety mechanisms have to be provided by the execution environment (e.g. Memory Protection, Watchdog, Time and Execution Protection). If no external protection mechanisms for the memory of the E2EPM exists then the E2EPM provides a redundant implementation variant for the self-protection of E2EPM internal data. This safety mechanisms assures freedom from interference of the E2EPM regarding data corruption.

[EB_E2ESE003090]

The main features provided by the self protection mechanism are:

- Time diversity: The self protection mechanism provides redundant send functions on the sender side and redundant receive functions on the receiver side.
- Data redundancy: The redundant routines internally use redundant data representations (e.g. status information, configuration data) except for the application data element, because the application data element is instantiated only once. In order to detect faults related to the application data element of the SW-C (e.g. random memory corruption), the data element is written/read by the application at each call of a redundant wrapper routine.

[EB_E2ESE003100]

The data and control flow for the E2EPM Communication Self Protection Mechanism is depicted in Figure 3.4, “Data flow and control flow for the Self Protection Mechanism”.

Legend: The control flow is depicted with solid arrows in red color and the data flow with dotted arrows in blue color. Oval shapes in red represent control flow actions. Solid boxes in blue represent persistent data. Dotted boxes in blue represent intermediate data.

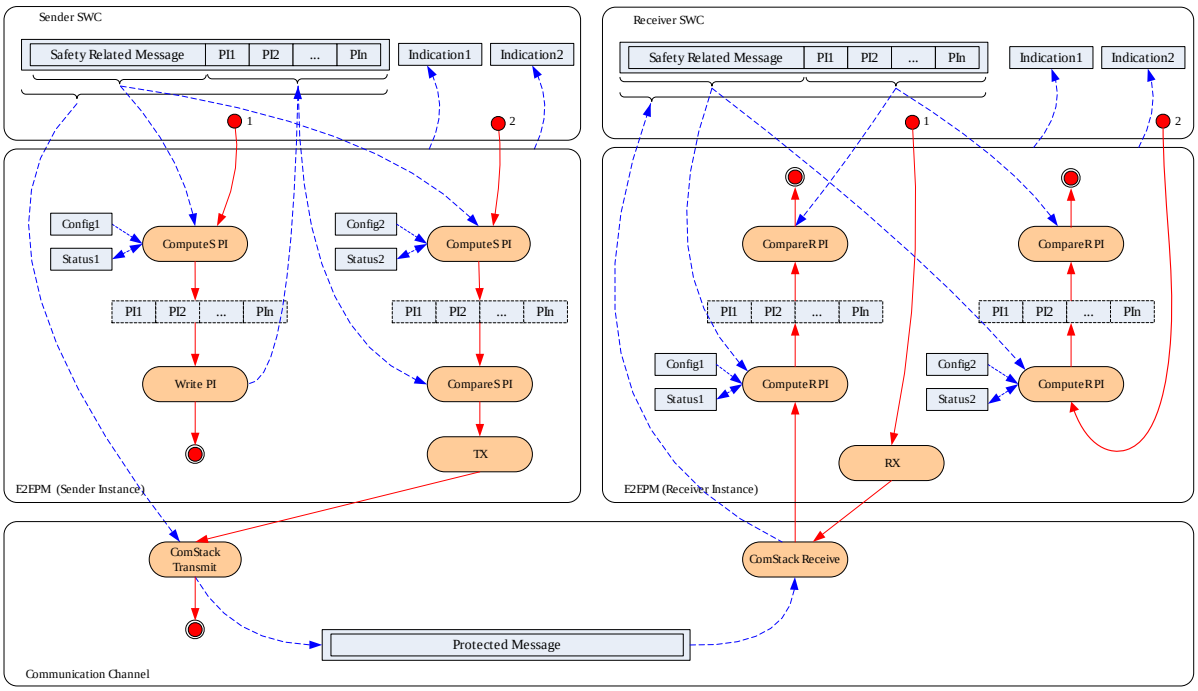

Figure 3.4. Data flow and control flow for the Self Protection Mechanism

[EB_E2ESE003110]

Components and their data structures:

- Sender SW-C
  - Safety Related Message: contains safety related data that should be transmitted over the communication channel
  - (PI1, ..., PIn): Protection Information (PI) (e.g. CRC, Sequence/Alive Counter, unique data ID)
  - Indication1,Indication2: indication from channel 1 / 2 of an application error (Range check) as well as internal errors of the E2EPM, or external errors
- E2EPM (Sender Instance)
  - Config1, Config2: redundant representation for data ID
  - Status1, Status2: redundant representation for last sent sequence counter
  - PI: locally computed PI
- Communication Channel
  - Protected Message: contains the safety related Message with added PI
- E2EPM (Receiver Instance)
  - Config1, Config2: redundant representation for data ID and further Rx properties (e.g. MaxDeltaCounterInit, MaxNoNewOrRepeatedData, SyncCounterInit, etc.)

- Status1, Status2: redundant representation for last successfully received sequence counter and additional information for error detection (e.g. DeltaCounter, number of repeated data counter, sync counter, etc.)
- PI: locally computed PI

- Receiver SW-C
  - Safety Related Message: contains the safety related data that was received over the communication channel
  - (PI1, ..., PIn): PI (PI) of the safety related data that was received (e.g. CRC, Sequence/Alive Counter, unique data ID)
  - Indication1, Indication2: indication from channel 1 / 2 of a communication error as well as internal errors of the E2EPM, or external errors

[EB_E2ESE003120]

Control Flow at the Sender SW-C:

- Sender SWC initiates the sending of the safety related payload.
- 1st control flow
  - ComputeS PI: computes the PI based on
    - the safety related message,
    - the internal configuration Config1 (e.g. data ID),
    - and the status information Status1 (e.g sequence counter)
  - Write PI: writes the locally computed PI to the data container of the PI provided by the Sender-SWC.
  - Result Indication: Indication of an application error (e.g range check) as well as internal errors to the Sender-SWC.
- 2nd control flow
  - ComputeS PI: computes the PI based on
    - the safety related message,
    - the internal configuration Config2 (e.g. data ID),
    - and the status information Status2 (e.g sequence counter)
  - CompareS PI: compare the previously computed PI with the newly computed PI and may indicate a difference
  - TX: the E2EPM passes the message to ComStack Transmit
  - ComStack Transmit: Transmission of the serialized data
  - Result Indication: Indication of an application error (e.g range check) as well as internal errors to the Sender-SWC.

[EB_E2ESE003130]

Control Flow at the Receiver SW-C:

- Receiver SW-C: cyclically initiates the reception of a safety related message
- 1st control flow
  - RX: the SW-C initiates the E2EPM to pass the receive request to the ComStack Receive function.
  - ComStack Receive: copies the received protected message to the data container in the receiver SWC
  - ComputeR PI: computes the expected PI based on
    - the safety related message,
    - the internal configuration (e.g. data ID),
    - and the status information (e.g last received sequence counter)
  - CompareR PI: compares the received PI with the expected PI.
  - Result Indication: Indicates the result (communication error, ..) to the receiver SW-C
- 2nd control flow
  - ComputeR PI: computes the expected PI based on
    - the safety related message,
    - the internal configuration (e.g. data ID),
    - and the status information (e.g last received sequence counter)
  - CompareR PI: compares the received PI with the expected PI
  - Result Indication: Indicates the result (communication error, ..) to the receiver SW-C

3.2.4.1.4. Interaction of Provided Safety Mechanism

Basically there is no dependency between Redundant Wrapper and E2E Communication Protection Mechanism.

3.2.5. Interfaces and Interaction of the EB tresos Safety E2E Wrapper

3.2.5.1. Interfaces of the E2EPM (Static view)

[EB_E2ESE003140]

Figure 3.5, “E2EPM Overview” depicts the E2EPM and its interfaces to the item (surrounding system):

- E2EPW API: interface to the application that uses the E2EPM for the protected communication (send and receive) of signals.
- Communication Description: the Communication Description holds the information about the network, the communication matrix, and the protection of signals.
- E2EPM User: the E2EPM user interacts with E2EPM in various ways: as system designer, as SW-C developer, or as integrator who configures the E2EPM.
- ASR RTE API: interface to the ASR communication stack. The Run Time Environment (RTE) constitutes this interface.
- Header files: the E2EPM uses header files provided by the AUTOSAR standard types and further platform integration files.

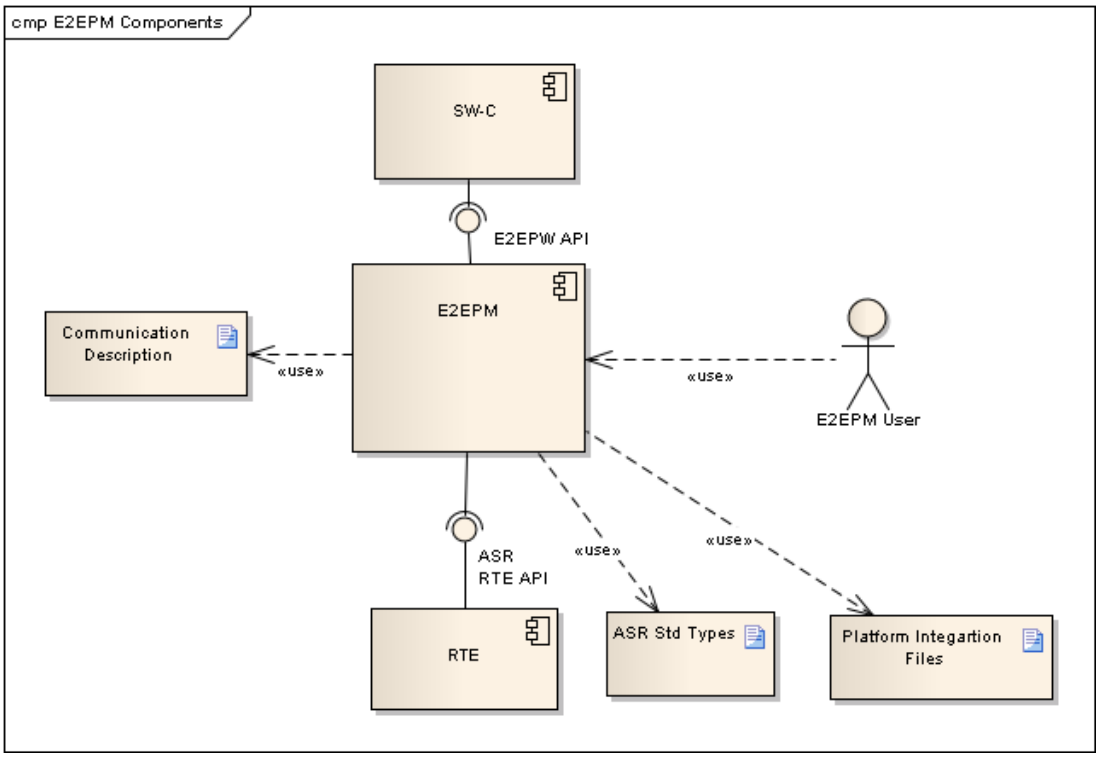

Figure 3.5. E2EPM Overview

3.2.5.2. Interaction of the E2EPM (Dynamic view)

[EB_E2ESE003150]

Figure 3.6, “E2EPM Dynamic Overview of Single Channel Wrapper” presents the dynamic behavior if a safe communication is established between SW-Cs via E2EPM by the use of single channel wrappers which requires no change of the application logic of the SW-Cs. Therein, the Sender SW-C transmits (usually periodically) a message to the Communication Link (i.e. the RTE) by calling `E2EPW_Write()`

instead of `Rte_Write()`. At the receiver side, the SW-C invokes `E2EPW_Read()` instead of `Rte_Read()` to perform a non-blocking read for receiving the message. Based on the type of reception trigger (event-driven or state-driven), different concepts must be established in order to detect timeouts at the receiver: If the receiver is event-driven (i.e. receiver is triggered on the message reception), then the receiver shall implement its own timeout detection mechanism (is not part of the E2EPM). If the receiver is state-driven (i.e. receiver is triggered independently of the data reception event - for example periodically triggered), then timeout detection is ensured by evaluating the counter (i.e. the failure mode message repetition is detected).

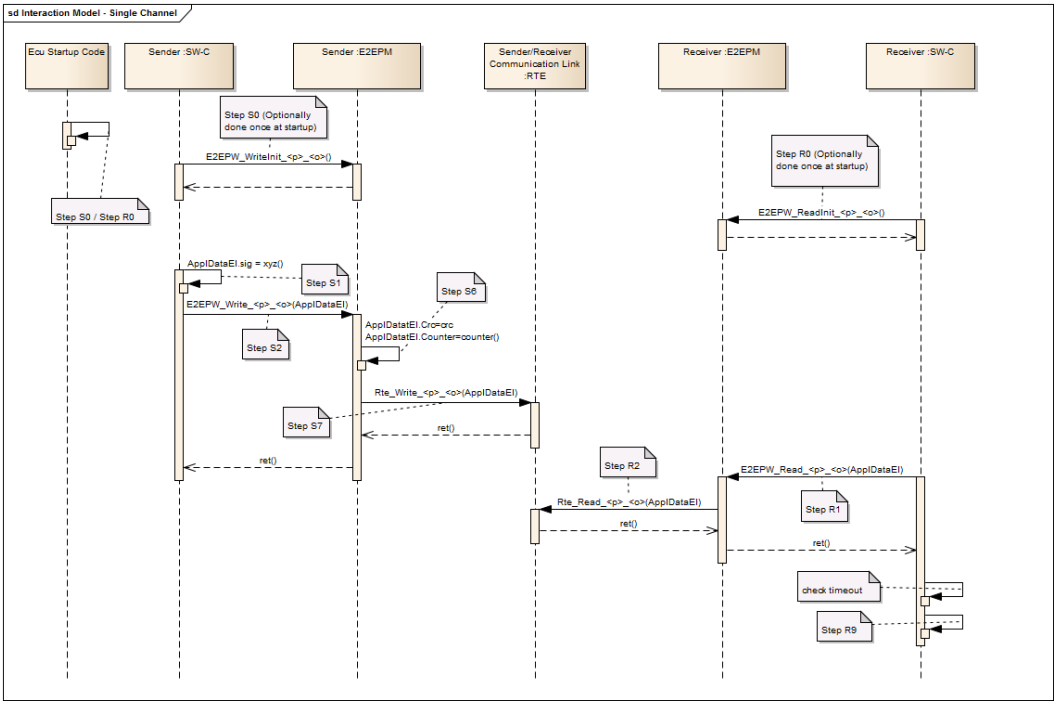

Figure 3.6. E2EPM Dynamic Overview of Single Channel Wrapper

[EB_E2ESE003303]

Figure 3.7, “E2EPM Dynamic Overview of Redundant Channel Wrapper” presents the dynamic behavior if a safe communication is established via E2EPM by the use of redundant channel wrappers which requires an adaptation of the application logic of the SW-Cs. In contrast to the single channel wrapper concept, the sender and receiver SW-C have to invoke the wrappers twice (Write1, Write2, Read1, and Read2). Because two different outputs (one from Write1 and one from Write2) are generated, therefore they are voted by Write2, before sending them through RTE. Only Write2 passes the data to the RTE in case of a successful voting the control fields. On the receiver side, only Read1 takes the message from the Communication Link (i.e. RTE). The SW-C has to check and compare the results of both Read1 and Read2.

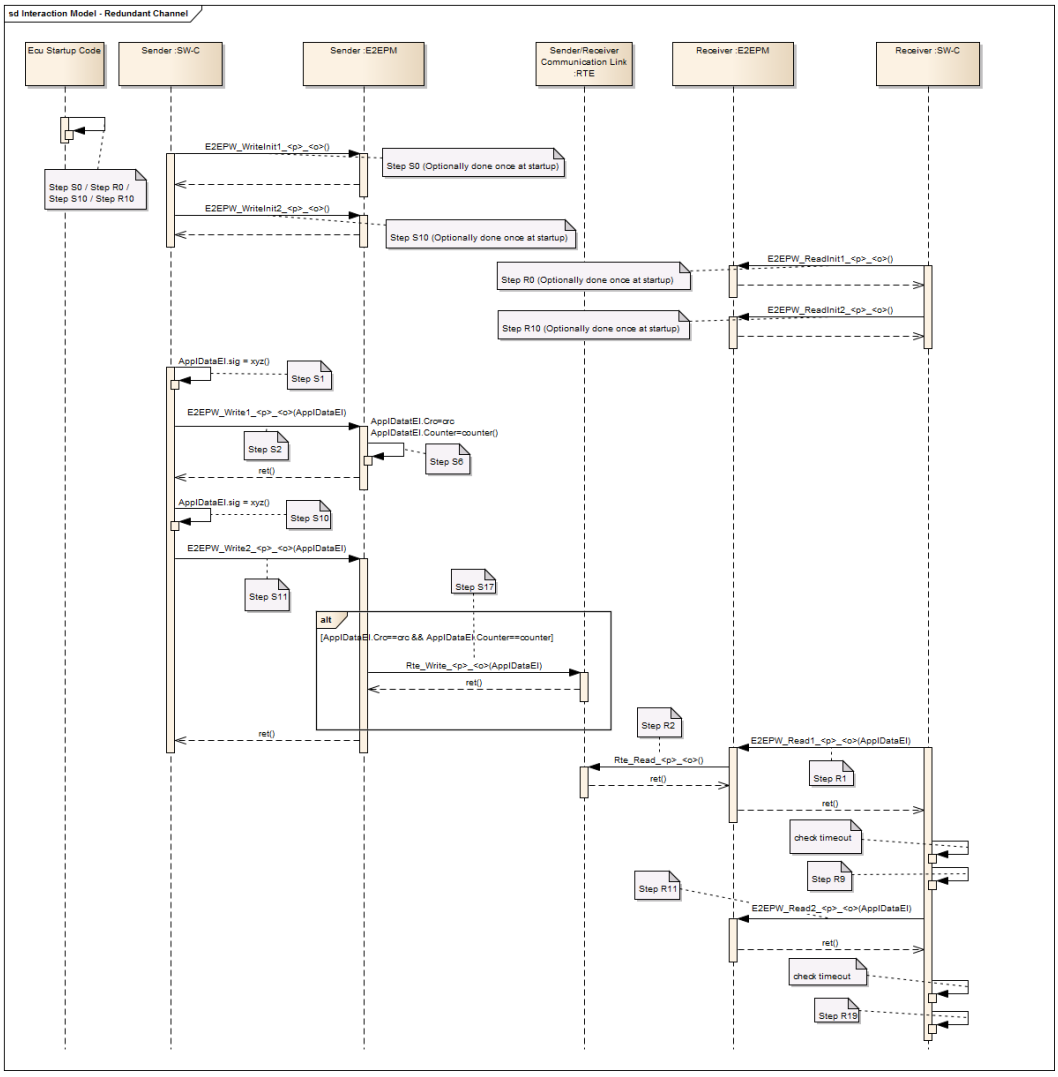

Figure 3.7. E2EPM Dynamic Overview of Redundant Channel Wrapper

3.2.5.3. Interaction with Item Safety Mechanisms

The E2EPM does not provide interfaces to the safety mechanisms of the item. It is assumed that depending on the ASIL of the data sink or data source the item implements adequate safety mechanisms like control flow monitoring or memory protection in a way that they do not interact with the E2EPM.

3.2.5.4. E2EPM Interaction with Specific Software

The E2EPM has interfaces to the RTE. The E2EPM is typically called from an SW-C and is therefore executed in this context.

3.2.6. Structure of the EB tresos Safety E2E Wrapper

The functionality of the complete E2EPM is decomposed into several units located either on a PC or on the ECU (Figure 3.8, “E2EPM Decomposition and deployment”).

Components / Artefacts on the PC:

EB tresos Studio (is no direct E2EPM component):

EB tresos Studio provides the user with a GUI for administration and controlling of the E2EPM Target SW components.

EB tresos Studio is invoked by the Integrator and provides an interface to import the AUTOSAR Communication Description into an internal database which can be accessed by EB tresos Studio Public APIs.

EB tresos Studio is classified with a Tool Confidence Level of TCL-1 (TCL-1 is given only for import and execution of the E2EPM Config Plugins) due to the fact that the generated code is verified by the E2EPW Check tool.

- E2EPM Config Plugins:
  - Configurable E2E Rx properties Wizard: updates and fills the E2EPW Config based on the information of the imported input data.
  - E2EPW MCG: the E2EPW MCG generates the E2EPW module based on the internally stored Communication Description within EB tresos Studio.
- E2EPW Check: this tool checks the E2EPW (generated) regarding its correctness and additionally reports manual review instructions for the user. If the user can provide a reliable Communication Description the E2EPW Check tool can optionally use this file to further automate the correctness check. The results of this check are stored in a Verification Report. E2EPW Check is classified with a Tool Confidence Level of TCL-3.
- E2EPW Config: holds additional information required for the E2EPW MCG which are not represented in the input data or shall overrule E2E configuration parameter.
- Verification Report: report of the check between E2EPW (generated) and the information stored in the files Communication Description and E2EPW Config.
- E2EPM Documentation: user manual and safety manual describing the tools, workflow, and guidelines of the E2EPM.

Components on the ECU:

- SW-C: user application that uses the signal protection mechanisms of the E2EPM Target SW.
- E2EPM Target SW
  - E2EPW Target SW
    - E2EPW (generated): provides the interface of the E2EPM to the user application. This software part is automatically generated by the E2EPW MCG.

- E2EPW (static): static code part of the E2EPW Target SW.

- E2E Libraries: consists of a static source code which provides a set of protection mechanisms which are implemented as re-entrant, stateless, and non-configurable functions according to ASR SWS [12].
- SCrc: a SCrc libbrary that consists of a static source code which provides re-entrant, stateless, and non-configurable CRC functions for E2E protection.

- RTE: provides the application interface of the Run Time Environment (RTE) for the communication of signals.

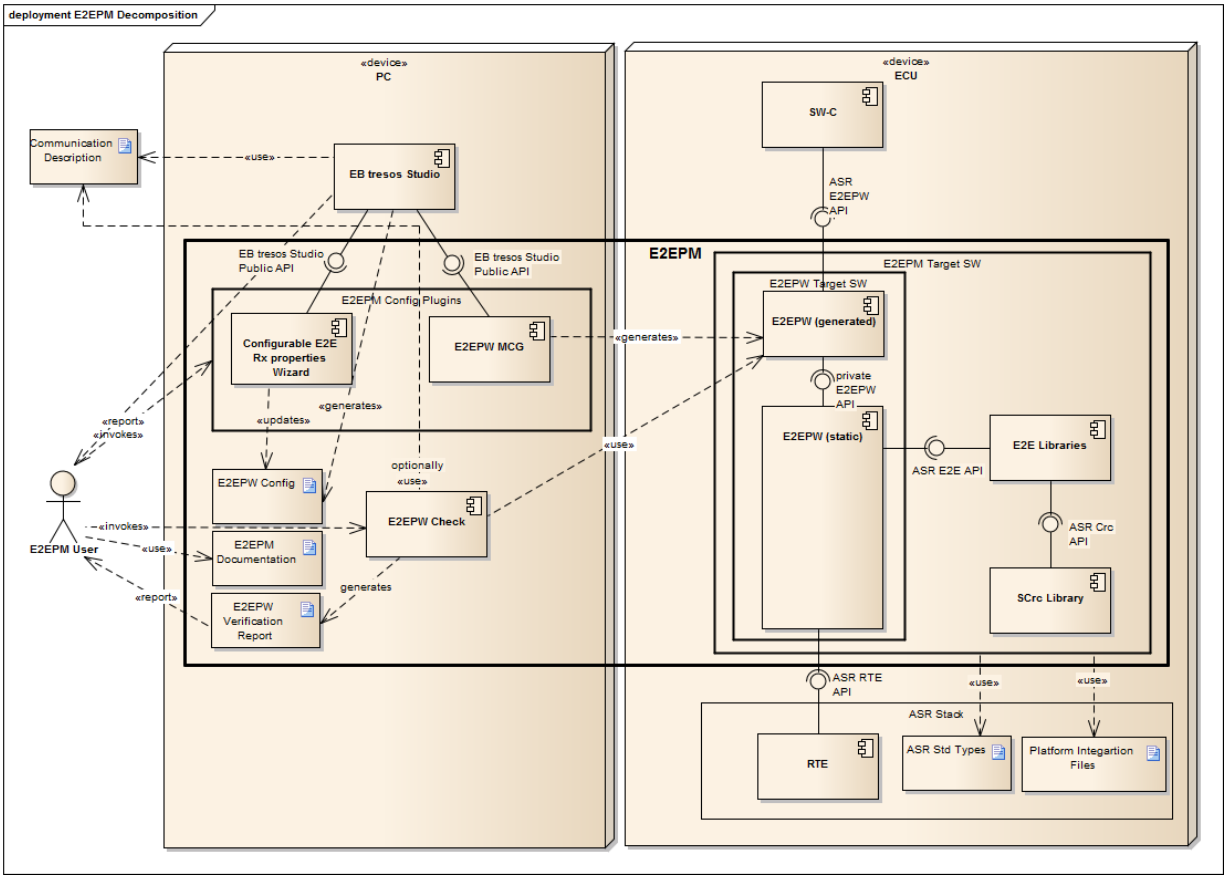

Figure 3.8. E2EPM Decomposition and deployment

[EB_E2ESE060003]

The E2EPM consists of several units as described in the E2EPM safety manual. In case only parts of the E2EPM are used (which is NOT recommended), the user shall ensure that (1) the unused parts are replaced by an alternative implementation according to the specific use case, and (2) the resulting software for E2E Protection (i.e. used E2EPM parts and users software) correctly implements the safety requirements.
Note:

3.2.7. Dynamic Behavior of the EB tresos Safety E2E Wrapper

3.2.7.1. Dynamic Behavior

The description of the dynamic behavior of the E2EPW is distinguished between the sender and receiver side and consists of several steps as documented in [ASRSWSE2E]. These steps are re-introduced hereinafter for both the single channel and redundant channel wrapper routines. The presented sequence diagrams shall only visualize the basic interactions between the E2EPM components on the ECU. Therefore, not all steps are visualized.

3.2.7.1.1. Sender SW-C (Single channel wrapper and first channel of redundant wrapper)

[EB_E2ESE003170]

Figure 3.9, “E2EPM Dynamic view of the Single Channel Write Wrapper” presents the sequence diagram for calling the single channel wrappers. Steps S0 to S9 apply also to Figure 3.10, “E2EPM Dynamic view of the Redundant Channel Write Wrappers” but `Write` is replaced with `Write1`.

- **Step S0:** Initialization of all necessary data structures. This is usually done either in the startup-code of the ECU or by a previous call to `E2EPW_WriteInit_<p>_<o>()`.

  Note: In case of Redundant Channels, a call to `E2EPW_WriteInit1_<p>_<o>()` and `E2EPW_WriteInit2_<p>_<o>()` ensures synchronous counter values for both channels.
- **Step S1:** The SW-C writes the values to the data element named e.g. `Appl1DataEl`.
- **Step S2:** The SW-C calls `E2EPW_Write_<p>_<o>()` instead of `Rte_Write_<p>_<o>()` and passes the data element.
- **Step S3:** The E2EPW serializes the data element to the local byte array with the layout of the corresponding signal group in the I-PDU and performs range checks on the values of the data element members.
- **Step S4:** The E2EPW calls the E2E library `E2E_PXXProtect()` in order to calculate the protection information.
- **Step S5:** The E2E library calls SCrc library in order to calculate the CRC value.
- **Step S6:** The E2EPW updates the protection information of the data element.
- **Step S7:** The E2EPW calls `Rte_Write_<p>_<o>()`.
- **Step S8:** The E2EPW calculates the return value and returns.
- **Step S9:** The SW-C checks the return value and executes an appropriate error handler which is specific to the application.

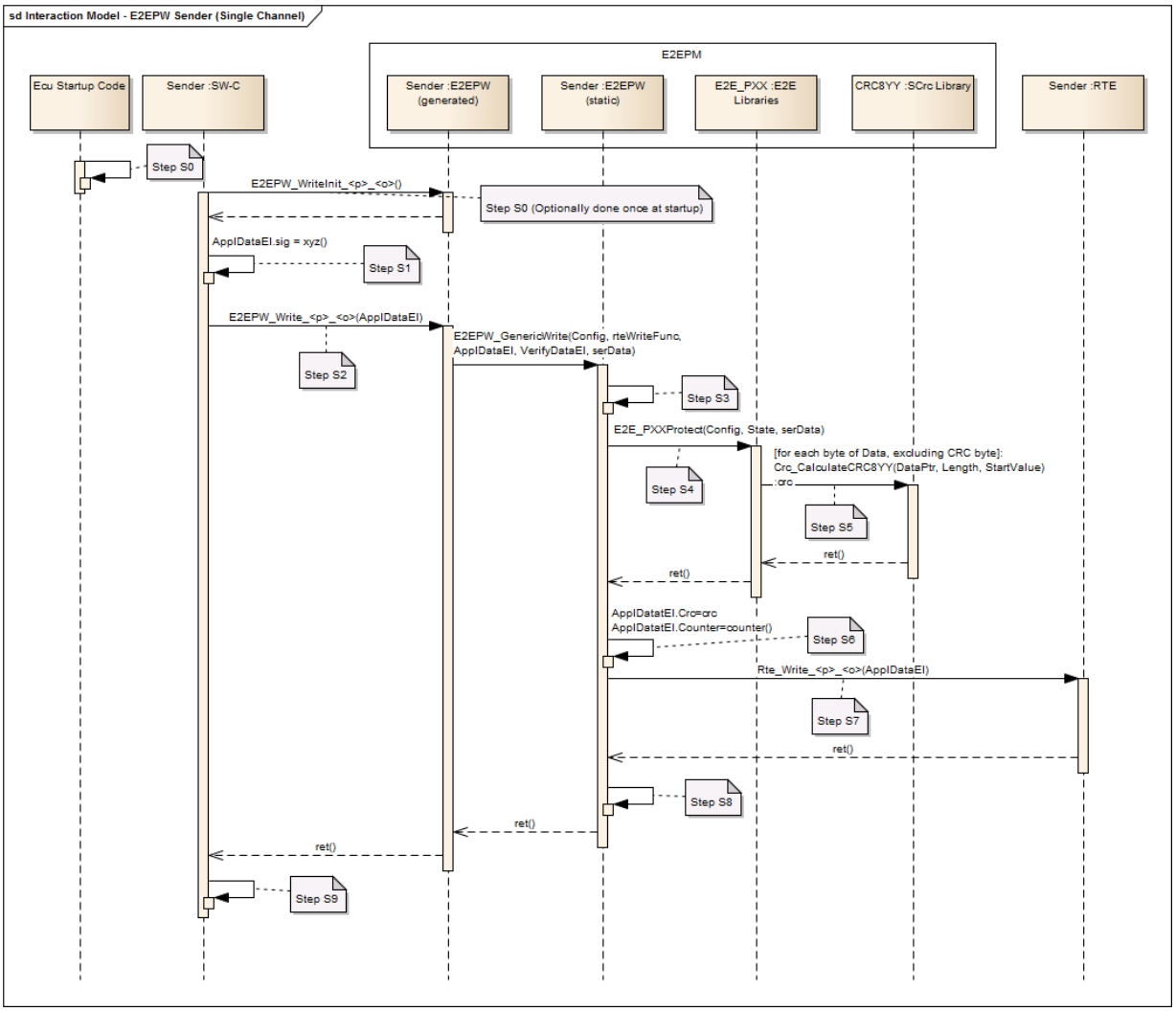

Figure 3.9. E2EPM Dynamic view of the Single Channel Write Wrapper

3.2.7.1.2. Sender SW-C (Redundant channel wrapper)

[EB_E2ESE003180]

Figure 3.10, “E2EPM Dynamic view of the Redundant Channel Write Wrappers” presents the sequence diagram for calling the redundant channel wrappers. The redundant channel variant is useful to detect faults in the memory used by the E2EPW (e.g. configuration and state variables are corrupted). The usage is similar to that of the single channel variant except that Step S0 (initialization) initializes Redundant Channel 1 wrapper data, Step S7 (call of Rte_Write) is skipped and Step S2 calls `E2EPW_Write1_<p>_<o>()` instead of `E2EPW_Write_<p>_<o>()`. Thus, only the additional steps following S9 are explained afterwards.

- **Step S10:** Handle this Step consistent to Step S0 for Redundant Channel 2 Wrapper data.

- **Step S11:** The SW-C writes the application data as in Step S1 again to the same data element, named e.g. `ApplDataEl`.
- **Step S12:** The SW-C calls `E2EPW_Write2_<p>_<o>()` and passes the data element.
- **Step S13-S15:** These steps are equal to S3-S5.
- **Step S16:** Skipped (In contrast to Redundant Channel 1 write, protection information e.g. CRC or counter are not updated within the application data element).
- **Step S17:** The E2EPW votes on the equality of the protection information (CRC and sequence counter) between passed data element `ApplDataEl` (i.e. previously computed in `E2EPW_Write1_<p>_<o>()`) and the locally computed values. Only if protection information calculated from Redundant Channel 1 and 2 are equal, then the data element is transmitted.
- **Step S18:** The E2EPW calculates the return value and returns.
- **Step S19:** The SW-C checks the return value and executes an appropriate error handler which is specific to the application. The SW-C additionally compares the return values of `E2EPW_Write1_<p>_<o>()` and `E2EPW_Write2_<p>_<o>()`.

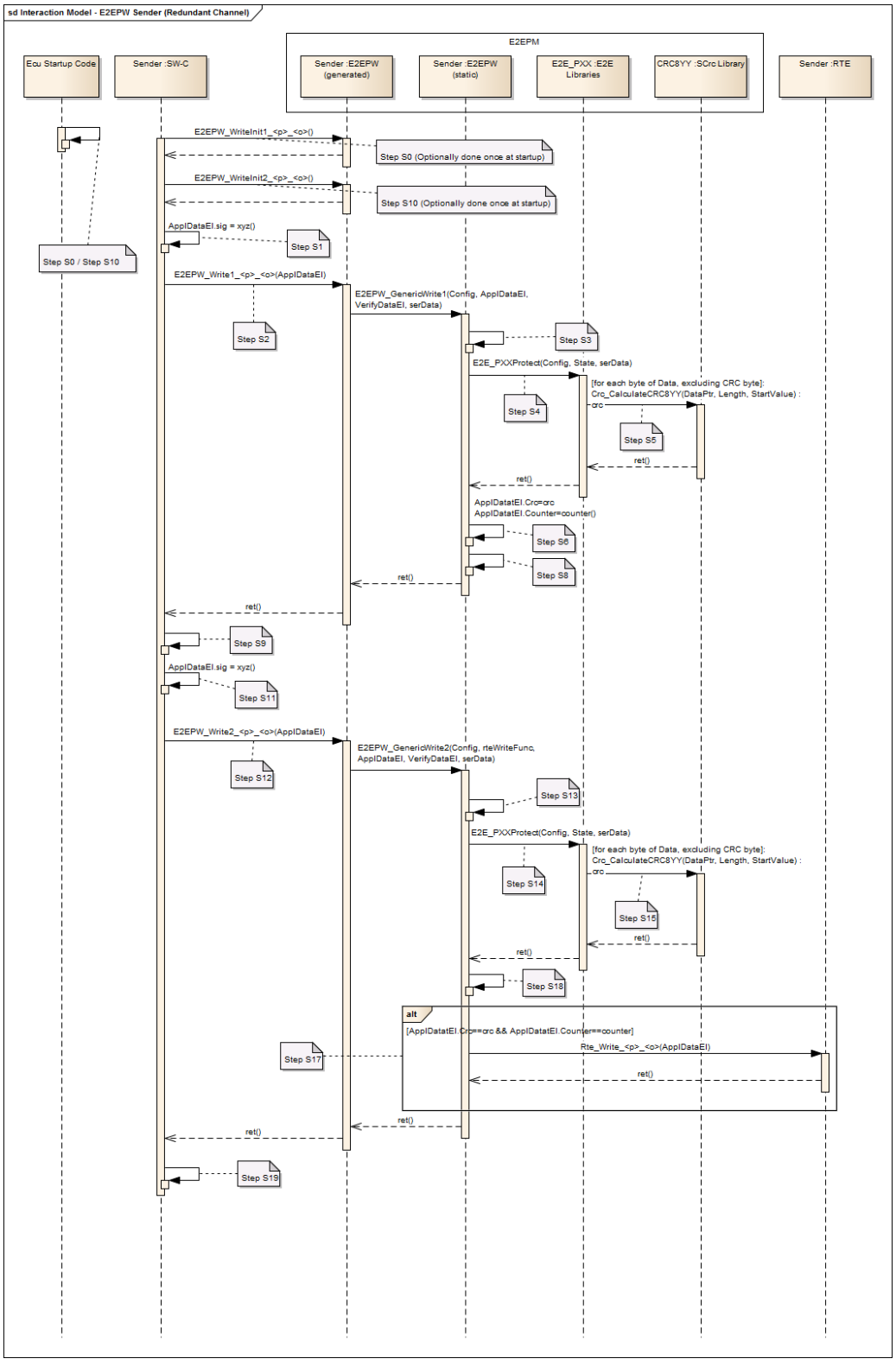

Figure 3.10. E2EPM Dynamic view of the Redundant Channel Write Wrappers

3.2.7.1.3. Receiver SW-C (Single channel wrapper and first channel of redundant channel wrapper)

[EB_E2ESE003190]

Figure 3.11, “E2EPM Dynamic view of the Single Channel Read Wrapper” presents the sequence diagram for calling the single channel wrappers. Steps S0 to S9 apply also to Figure 3.12, “E2EPM Dynamic view of the Redundant Channel Read Wrappers” but `Read` is replaced with `Read1`.

- **Step R0:** The E2EPW initializes all necessary data structures. This is usually done either in the start-up-code of the Ecu or a previous call to `E2EPW_ReadInit_<p>_<o>()`.
- **Step R1:** The SW-C calls `E2EPW_Read_<p>_<o>()` instead of `Rte_Read_<p>_<o>()` and passes the data element. Note: In case of Redundant Channels, a call to `E2EPW_ReadInit1_<p>_<o>()` and `E2EPW_ReadInit2_<p>_<o>()` ensures synchronous counter values for both channels.
- **Step R2:** The E2EPW calls `Rte_Read_<p>_<o>()` to receive the data element. Optionally the RTE function `Rte_IsUpdated()` is used to check if new data are available.
- **Step R3:** The E2EPW serializes the data element to the local byte array.
- **Step R4:** The E2EPW calls the E2E library `E2E_PXXCheck()` to detect possible communication errors.
- **Step R5:** The E2E library calculates the crc via calling the SCrc library, checks the data element against communication errors.
- **Step R6:** Skipped (In contrast to E2EPW_Write, protection information is not modified / updated within the application data element).
- **Step R7:** The E2EPW performs range checks on the values of the data element members.
- **Step R8:** The E2EPW calculates the return value and returns .
- **Step R9:** The SW-C performs deadline monitoring to detect reception timeouts and checks the return values. In case of successful return values, the application makes use of the application data. Otherwise, the application executes an appropriate error handler.

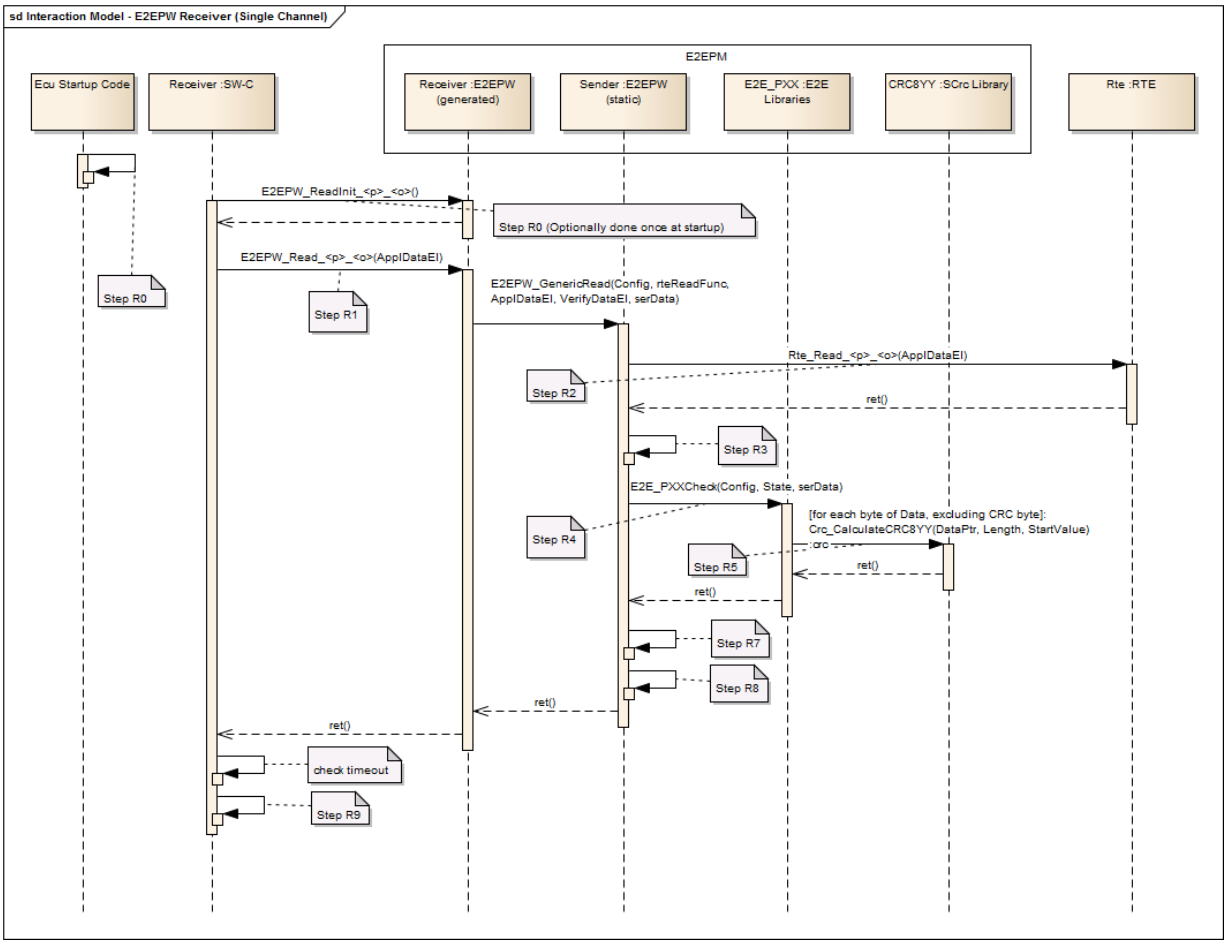

Figure 3.11. E2EPM Dynamic view of the Single Channel Read Wrapper

3.2.7.1.4. Receiver SW-C (Redundant channel wrapper)

[EB_E2ESE003200]

Figure 3.12, “E2EPM Dynamic view of the Redundant Channel Read Wrappers” presents the sequence diagram for calling the redundant channel wrappers. The redundant channel variant is useful to detect faults in the memory used by the E2EPW (e.g. config and state variables are corrupted). The usage is similar to that of the single channel variant except that in Step R0 (initialization) initializes Redundant Channel 1 wrapper data and Step R1 calls `E2EPW_Read1_<p>_<o>()` instead of `E2EPW_Read_<p>_<o>()`. Thus, only the additional steps following R9 are explained afterwards.

- **Step R10:** Handle this Step consistent to Step R0 for Redundant Channel 2 Wrapper data.
- **Step R11:** The SW-C calls `E2EPW_Read2_<p>_<o>()` and passes the same data element as stated in step R1.
- **Step R12-R18:** R12 is skipped (data are used from channel 1 passed in step R11) R13-R18 equal to R3-R8.

- **Step R19:** The SW-C performs deadline monitoring to detect reception timeouts and checks the return values. In case of successful return values, the SW-C additionally compares the return values of `E2EPW_Read1_<p>_<o>()` and `E2EPW_Read2_<p>_<o>()`. If equal, the SW-C further compares against the equality of the content of the data element after calling `E2EPW_Read1_<p>_<o>()` and `E2EPW_Read2_<p>_<o>()`. If all aforementioned steps are successful, then the application makes use of the application data. Otherwise, the application executes an appropriate error handler.

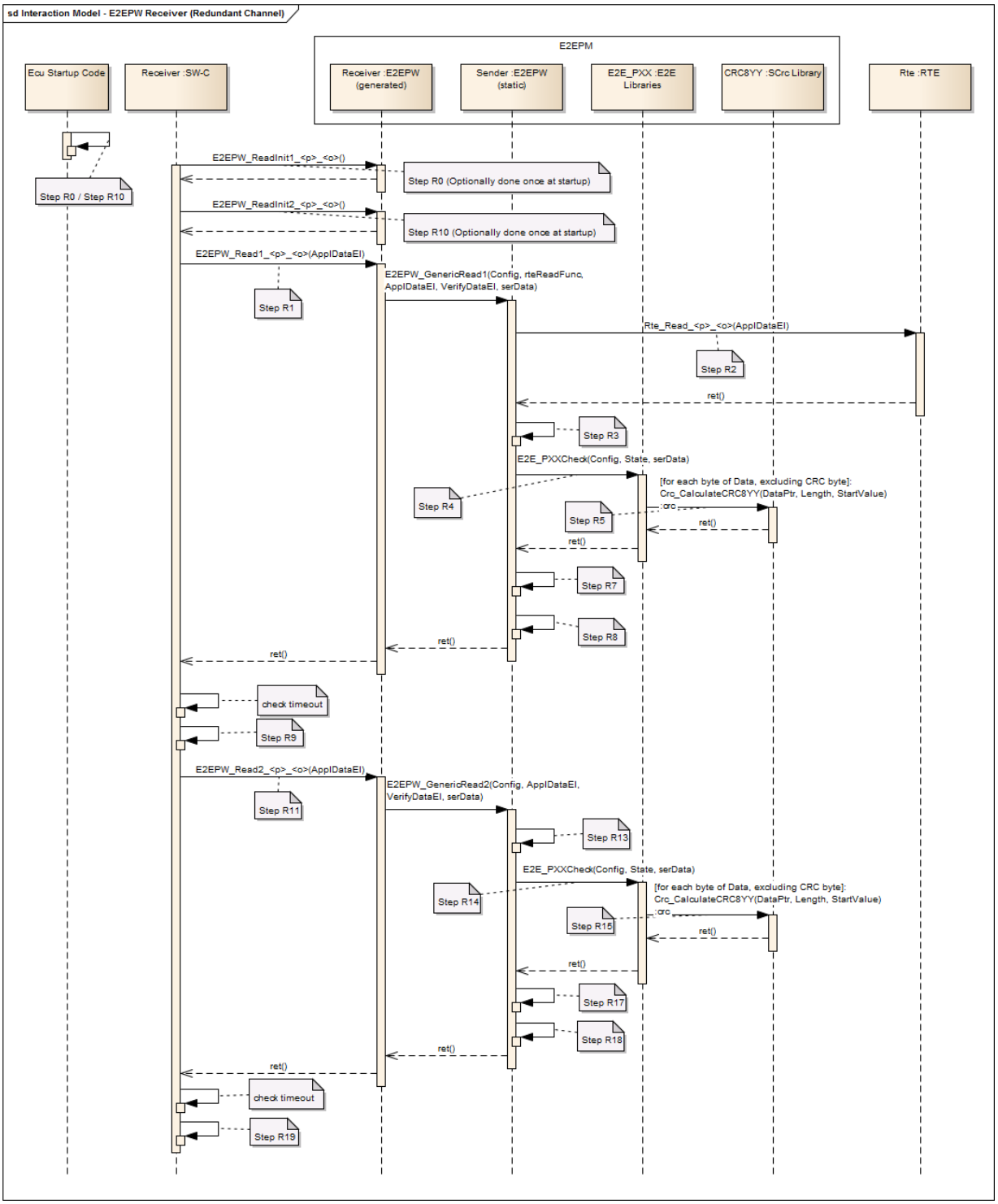

Figure 3.12. E2EPM Dynamic view of the Redundant Channel Read Wrappers

3.2.8. Configuration of the EB tresos Safety E2E Wrapper

3.2.8.1. E2EPM Configuration

3.2.8.1.1. Configuration Data

As described in chapter Section 3.2.6, “Structure of the EB tresos Safety E2E Wrapper”, the E2EPM consists of a static and a generated part. The generated part ("E2EPW (generated)", see Figure 3.8, “E2EPM Decomposition and deployment”) is application specific and represents the configuration data.

The generated part consists of the following files:

- `E2EPW_Int_Cfg.h`: This file is generated once for a ECU and contains pre-compile time configuration parameters (i.e. macro definitions).
- `E2EPW_<SWCShortname>.h`: This H-file is generated for each SW-C with shortname <SWCShortname> which contains at least one protected data element. It contains configuration data in the form of function declarations for each protected Port/DataElement.
- `E2EPW_<SWCShortname>.c`: This C-file is generated for each SW-C with shortname <SWCShortname> which contains at least one protected data element. It contains constant configuration data which represents the E2E Description Information and function definitions for each protected port data element.

3.2.8.1.2. Calibration Data

The E2EPM has no calibration data.

3.2.9. Shared Resources of the EB tresos Safety E2E Wrapper

3.2.9.1. E2EPM Shared Resources

The E2EPM provides two software libraries the E2E library and the SCrc library. These libraries do not share any variables. The module E2EPW uses global variables that are only used by the SW-C specific code. All E2EPW functions are not reentrant for the same port data element and all E2EPW functions related to the same port data element shall not interrupt each other.

3.2.10. Operating Modes and Status Information of the EB tresos Safety E2E Wrapper

3.2.10.1. E2EPM Operating Modes and Status Information

3.2.10.1.1. Operating Modes

The E2EPM itself provides no operating modes. An example for a possible mode management that can be implemented by the user in the receiver based on the E2EPM is given in [ASRSWSE2E] in the Chapter 12 Annex B: Application hints on usage of E2E library. In the Figure 3.13, “Example for a mode management implemented in the receiver SW-C.” the corresponding return values of the E2EPM for implementing such a suggested state machine are presented.

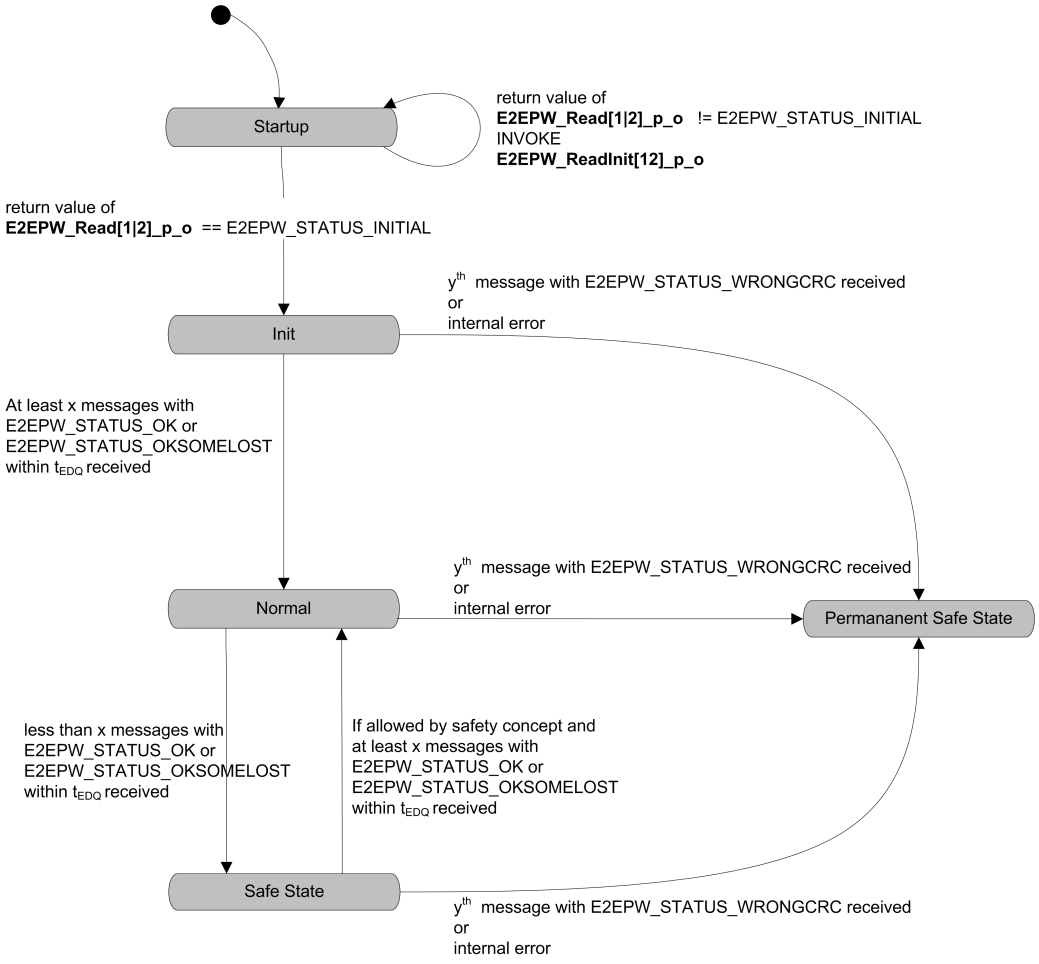

Figure 3.13. Example for a mode management implemented in the receiver SW-C.

Note that the actual values in this state machine for the time interval for Error Detection and Qualification (TEDQ), x (the number messages with E2EPW_STATUS_OK or E2EPW_STATUS_OKSOMELOST), and y (the number of messages with E2EPW_STATUS_WRONGCRC) need to be specifically computed by the user of the E2EPM.

3.2.10.1.2. Status Information

Status Information is provided via a 4-byte return value of the E2EPW Read/Write function which contains the following information:

- result of Rte call
- result of internal checks
- result of applied safety mechanism (protect/check function)

3.2.10.1.3. Safe State

The E2EPM is an SEooC and thus itself does not have a safe state, but it may contribute to a safety mechanism of the whole item that provides a safe state.

3.2.10.1.4. Malfunction Indication and Handling

Detected malfunction of the E2EPM is indicated via Byte 1 (value not equal E2E_E_OK) of the E2EPW Read/Write function return value.

If a malfunction is indicated, then E2EPM has to be considered as dis-functional. The user has to implement appropriate measures for handling the malfunction. In the case further operation is required the user shall restart the E2EPM by using E2EPW_ReadInit/E2EPW_WriteInit.

3.2.11. E2EPM Error Detection and Handling of the EB tresos Safety E2E Wrapper

3.2.11.1. E2EPM Error Detection and Handling

The E2EPM indicates transmission errors detected by the safety mechanism via the return value of the E2EPW Read function. This return value has to be evaluated by the SW-C.

3.2.12. Limitations of the EB tresos Safety E2E Wrapper

The limitation of the E2EPM are documented in the E2EPM Release Notes [EBASCE2ESE-24] [82]

3.2.13. Robustness of the EB tresos Safety E2E Wrapper

Describe the robustness against

- *Robustness against hardware faults*
- *Robustness against systematic software errors*
- *Robustness against configuration errors*
- *Robustness against resource conflicts*
- *Robustness against interrupt overload*
- *Robustness against input errors*
- *Robustness against data starvation*

3.2.14. What the EB tresos Safety E2E Wrapper does not do

Some users might want to solve problems with your EB tresos Safety E2E Wrapper that it was never intended for/is inappropriate for. Describe the limitations of your [Safety Product] (and what the user could do to solve such issues).

3.3. Outstanding anomalies

Details of all outstanding anomalies should be given, with explanation of the anomaly, how it occurs and the mechanisms that the integrator shall take to mitigate the anomaly should the particular functions be used.

3.4. Backward compatibility

Details of whether the element is compatible with previous releases of the subsystem, and if not, details of the process providing the upgrade path to be followed.

3.5. Change Control

The mechanism by which the integrator can initiate a change request to the producer of the software.

3.6. Assumptions of EB tresos Safety E2E Wrapper

This section defines the assumptions on the item.

3.6.1. Item Safety Requirements

The E2EPM has no specific safety requirements on the item, i.e. there are no additional safety requirements on the item compared to common SWCs.

Note: In a typical usage scenario the E2EPM will be integrated on one micro controller together with non safety related components. Here freedom from interference is required in order to allow coexistence of safety and non safety related components. For the protection of the E2EPM against failures from non safety related components the E2EPM provides a redundant wrapper implementation variant as self-protection mechanism to reduce the requirements on the execution environment in order to achieve freedom from interference regarding memory (see Section 3.1, “Architecture of the surrounding system”). Failures resulting from the E2EPM are prohibited by the development process of the E2EPM according to ASIL-D under the condition that the obligations documented in this safety manual are fulfilled.

3.6.2. Item SW Requirements

See related requirement of chapter Section 6.2.1, “Integration Rules and Notes” and Section 6.3.1, “Application Rules and Notes”.

3.6.3. Item HW Requirements

[EB_E2ESE070001]

The *EB tresos Safety E2E Wrapper* assumes that the hardware provides a reliable execution environment.
Note: In *EB tresos Safety E2E Wrapper* a reliable execution environment is defined as a microcontroller that provides a mechanism that prevents or detects non-systematic hardware faults, e.g. data corruption or incorrect execution of instructions. Examples are lockstep mode and ECC memory.

[EB_E2ESE065131]

The target MCU has to be identical to one of the reference platforms specified in [EBASCE2ESE-24] or has been approved separately by EB. If any other 16-bit or 32-bit target MCU is chosen, then the user has to perform additional verification measures that are adequate and sufficient to ensure that there are no faults of the E2EPM originated in the chosen target MCU.
Note: Fulfilled by the userguide.

[EB_E2ESE065132]

The item has to provide enough RAM/ROM/RunTime resources for E2EPM. For the calculation of the required resources the values from the reference platforms specified in [EBASCE2ESE-24] can be used as basis.

Reference values are measured with single channel wrapper for the Communication Description of Test Config 3 for atomicSWC6 and atomicSWC7 (see [EBASCE2ESE-33]) with the following configuration for DataElement DE2:

DataElement DE2 contains the following members:

- CRC_Signal (INTEGER-TYPE with uint8)
- SEQ_Counter (INTEGER-TYPE with uint8)
- SigSint8 (INTEGER-TYPE with sint8)
- SigSint16 (INTEGER-TYPE with sint16)
- SigBool (INTEGER-TYPE with boolean)
- SigString[0] (STRING-TYPE with 1 character of type uint8)
- SigArray[0].SigRec.SigUint16 (SigArray=ARRAY-TYPE with one entry of SigRec; SigRec=RECORD-TYPE with one record element SigUint16; SigUint16=INTEGER-TYPE with uint16)

DE2 maps byte-aligned to a SignalGroup of length 64 bit as follows:

- CRC_Signal maps to a GroupSignal with: bitpos=0, bitlength=8, endianess=Little Endian, type=uint8
- SEQ_Counter maps to a GroupSignal with: bitpos=8, bitlength=4, endianess=Little Endian, type=uint8
- SigSint8 maps to a GroupSignal with: bitpos=32, bitlength=8, endianess=Little Endian, type=sint8
- SigSint16 maps to a GroupSignal with: bitpos=40, bitlength=16, endianess=Little Endian, type=sint16
- SigBool maps to a GroupSignal with: bitpos=12, bitlength=1, endianess=Little Endian, type=boolean
- SigString[0] maps to a GroupSignal with: bitpos=56, bitlength=8, endianess=Little Endian, type=uint8
- SigArray[0].SigRec.SigUint16 maps to a GroupSignal with: bitpos=16, bitlength=16, endianess=Little Endian, type=uint16

Note: Fulfilled by the userguide.

3.6.4. Item SW Tool Requirements

[EB_E2ESE065141]

E2EPM Target SW is delivered in source code and requires an appropriate Build-Environment (i.e. Compiler, Linker) and Flash-Tool for applying the created binary to the target hardware.
Note: Fulfilled by the userguide.

[EB_E2ESE065142]

The compiler version and settings/options have to be identical to one of the reference platforms specified in [EBASCE2ESE-24] or has been approved separately by EB. If any other compiler or setting/options are chosen, then the user has to perform additional verification measures that are adequate and sufficient to ensure that there are no faults of the E2EPM caused by the chosen compiler or settings/options.

The used compiler/linker must at least issue a warning or error in the following cases:

- non existing include files (e.g. "error: E2EPW_AtomicSWC1.h: No such file or directory")
- usage of undefined/undeclared identifier (function, variable, macro, etc.) (e.g. "error: undefined reference to 'E2E_P02Protect'" (issued by the linker), or "error: 'E2E_P01ConfigType' undeclared (first use in this function)", or "error: E2EPW_CFG_DM_E2EDesc undeclared here (not in a function)"), NOTE: can be in a function or global variable definition.
- access to not declared members of a structure (e.g. "error: structure has no member named `E2EPW_CFG_CC_E2EDesc'")
- redefinition of a macro with different values (e.g. "warning: "E2EPW_CFG_DI_ComSignalGroup_Rx" redefined")
- same function/variable declared several times (e.g. "warning: redundant redeclaration of 'E2EPW_PACK_E2EDesc'")
- same function/variable defined several times (e.g. "error: redefinition of 'E2EPW_PACK_E2EDesc'")
- calling a function with incompatible argument types (e.g. "warning: passing argument 1 of 'Rte_Read_SWC_p_o' from incompatible pointer type")
- implicit declaration of a function (e.g. "warning: implicit declaration of function 'Rte_Write_Port1_DataElement1'")

Note: Fulfilled by the userguide and an explicit test-case check that compiler reports adequate errors / warnings in case of different fault-injections (see ASCE2ESE-466).

[EB_E2ESE065143]

The compiler does not report any errors. The compiler does not report any warnings or all generated warnings can be considered as acceptable (i.e. code is correct for the used application) after they have been analyzed with diligent care.
Note: Fulfilled by the userguide.

3.6.5. Communication Description Rules and Notes

In this section the rules and notes for the input data regarding the workflow step "Import" are described.

3.6.5.1. Communication Description Notes

[ASR_E2EPW020089]

In case E2EPM is used to protect Data Elements, Data Elements shall use the RTE receive mode "Explicit data read access" (i.e. it shall not use the RTE receive mode "Implicit data read access").
Note: Fulfilled by the userguide.

[EB_E2EPW020123]

If a data element is of complex type, then the protection information (CRC, counter, if used: DataIdNibble) shall be of the primitive data type uint8.
Note:

[EB_E2EPW020040]

The Communication Description shall be according to SWS Software Component Template [ASRSWSSW-CT] with the following modification:

- Value of category in EndToEndDescription

  Description:

  [EB_constr_1001] The attribute category of EndToEndDescription can have the following values:

  - NONE
  - PROFILE_01
  - PROFILE_02
  - ProfileVCC

  Comment:

  This is a modification of constraint [constr_1110] which allows the support of additional profiles.

3.6.5.2. Communication Description Rules

[ASR_E2EPW020292]

If the E2E Library is invoked from E2E Protection Wrapper (at the level of Data Elements), then multiple instantiation is not allowed. For an AUTOSAR software component which uses the E2E Protection Wrapper the value of the attribute supportsMultipleInstantiation of the SwcInternalBehavior shall be set to FALSE in the AUTOSAR software component description.
Note:

[ASR_E2EPW020224]

The communication shall be an explicit sender-receiver communication, in 1:1 and 1:N multiplicities.
Note: Fulfilled by the userguide.

[ASR_E2EPW020271]

E2E Com Callouts shall not be used for Data Elements which are protected by E2EPM.

Note: Fulfilled by the userguide.

[ASR_E2EPW020255]

If the E2E Library is invoked at the level of Data Elements and 1:N communication model is used and the Data Elements are sent using more than one I-PDU, then all these I-PDUs shall have the same layout.

[EB_E2EPW020164]

Rte data conversion is not supported with E2E protected communication. That is, if data conversion is required it has to be implemented by the SWC (e.g. as Conversion Manager SWC, see ASR E2E, chapter "Application scenario with E2E Manager and Conversion Manager"). See also E2EUSE274, E2EUSE275, E2EUSE272, E2EUSE273 in [ASRSWSE2E].

As a consequence, the compuMethods (can be referenced in I-Signals, DataMappings, Sender/Receiver-ComSpecs, etc.) specified in the Communication Description according to ASR 4.0 are ignored.

4. Using the EB tresos Safety E2E Wrapper safely

EB tresos Safety E2E Wrapper is developed as a safety element out of context (SEooC). Therefore, Elektrobit Automotive GmbH assumes that the environment meets particular requirements so that the E2EPM code behaves appropriately and safely.

Each release of *EB tresos Safety E2E Wrapper* is developed with and tested against a particular AutoCore version, which is specified within the E2EPM release notes.

For more information on intended usage and possible misuse of E2EPM, see the module user guides.

4.1. Prerequisites

The E2EPM requires the following modules:

- *EB tresos Studio*
- *System Description*

The product versions of EB tresos Studio that is supported by this release is documented in the E2EPM release notes.

The delivery contains the following files

- README.txt
- Quality_Safety_Statement.pdf
- <project>-<release>-<version>-<date>.zip

4.2. Installing the EB tresos Safety E2E Wrapper

Before you install E2EPM:

- Make sure that the correct EB tresos Studio version is already installed.

To install E2EPM follow the steps described in chapter Installing additional products to EB tresos Studio in the EB tresos installation guide which is part of the file which you have downloaded from EB Command.

NOTE **Install all packages**

You must install all packages from the file which you have downloaded.

EB delivers a software with the following activities:

- upload delivery: EB uploads the delivery to the EB command server
- notify customer: EB notifies the customer about an uploaded delivery to EB Command.

The customer gets informed about the availability of a delivery and has to perform the following activities:

- download delivery: The customer downloads the delivery from EB Command.
- install delivery: The customer installs the delivery
- check installation: The customer verifies the correctness of the installed items with the SHA-1 hashes, refer to Section 4.3, “Verifying the integrity of the EB tresos Safety E2E Wrapper sources”

In case that the delivery is not directly received from EB, but from an authorized distributor (e.g. an OEM with the right to issue sublicences), then the customer and/or distributor has to assure that the delivery process is not affected (no loss of information).

After installation with setup-exe from the `$TRESOS_BASE/` the following folders are created:

Let <TresosBase> be the installation path of EB tresos Studio. After installation with setup.exe to the base path <TresosBase>, the following directories/files will be created:

- <TresosBase>\doc\7.0_EB_tresos_Safety_E2E_Protection

  Contains the following documents regarding the EndtoEnd Protection:

  - Safety_E2E_Wrapper_release_notes.pdf
  - Safety_E2E_Wrapper_safety_manual.pdf
- <TresosBase>\plugins

  Contains all installed module plugins (e.g. modules SCrc, E2E, E2EP02, and E2EPW)
- <TresosBase>\plugins\SCrc_TS_TxDx<ModuleVersionNumber>

  The SCrc module contains only static non-configurable code in the folders include and src
- <TresosBase>\plugins\E2E_TS_TxDx<ModuleVersionNumber>

  The E2E module contains only static non-configurable code in the folders include and src
- <TresosBase>\plugins\E2EPXX_TS_TxDx<ModuleVersionNumber>

  The E2E Profile XX module contains only static non-configurable code for E2E Profile XX where XX can be one or more of 01, 01A, 01B, 02, 03, etc.

- <TresosBase>\plugins\E2EPW_TS_TxDx<ModuleVersionNumber>

  The E2EPW module contains only static non-configurable code in the folders include and src
- <TresosBase>\plugins\E2EPW_TS_TxDx<ModuleVersionNumber>\check\dist\E2EPWCheck.exe

  The executable of the tool E2EPW Check.
- <TresosBase>\plugins\E2EPWExt_TS_TxDx<ModuleVersionNumber>

  This module plugin is part of the E2EPW module and contains the module code generator
- <TresosBase>\templates

  This directory contains the sample project E2EProtection_Template
- <TresosBase>\workspace

  Will be created if EB tresos Studio is started the first time. This folder will later contain several projects created by the user.
- <TresosBase>\checksums

  Contains all SHA-1 hash files of a module.

4.3. Verifying the integrity of the EB tresos Safety E2E Wrapper sources

The E2EPM delivery contains the checksums of all files that are part of the shipped package. You can find these files in `$TRESOS_BASE/checksums` folder of each module.

Verify for each module that the SHA-1 hash of the file `<module><version>.sha1.sha1` (i.e. `E2EPW_TS_TxDxM1I0R0.sha1.sha1`) matches to the corresponding SHA-1 hash of the reference file <version>README.txt. The <version>README.txt file is delivered via EB command.

In case that the delivery is not directly received from EB, but from an authorized distributor (e.g. an OEM with the right to issue sublicences), then the customer and/or distributor has to assure that the delivery process is not affected (no loss of information).

WARNING **Use E2EPM only with correct files**

Use E2EPM only if all files are verified and intact.

In case of corrupted files, reinstall the package that contains E2EPM, and watch out for any warnings that appear during this process. Contact Elektrobit support if verification still fails.

4.4. Field Monitoring

Periodically a list of all related problem reports (known issue list) is automatically created for each delivery under maintenance and provided to the customer via the EB Command-Server. The list provides the following information:

- related release version.
- list of published bugs with the following information: unique Id of bug, affected module and version, defect description, and workaround (if applicable).

The access to the known issue list is protected by customer specific login and password (the Command login).

4.5. Correct use and integration of the EB tresos Safety E2E Wrapper

Describe how to use and integrate the EB tresos Safety E2E Wrapper. This can consist of:

- Describe how to do the configuration for safety-relevant parts
- Describe how the user can generate the code (if automatically generated parts are contained)
- Describe how the user can verify that automatically generated code is correct and safe

4.5.1. Workflow of the EB tresos Safety E2E Wrapper

[EB_E2ESE003210]

Figure 4.1, “E2EPM Workflow” shows the workflow and the different activities allocated to the participating systems for the configuration of the E2EPM and the generation of the specific E2E Protection Wrapper functions:

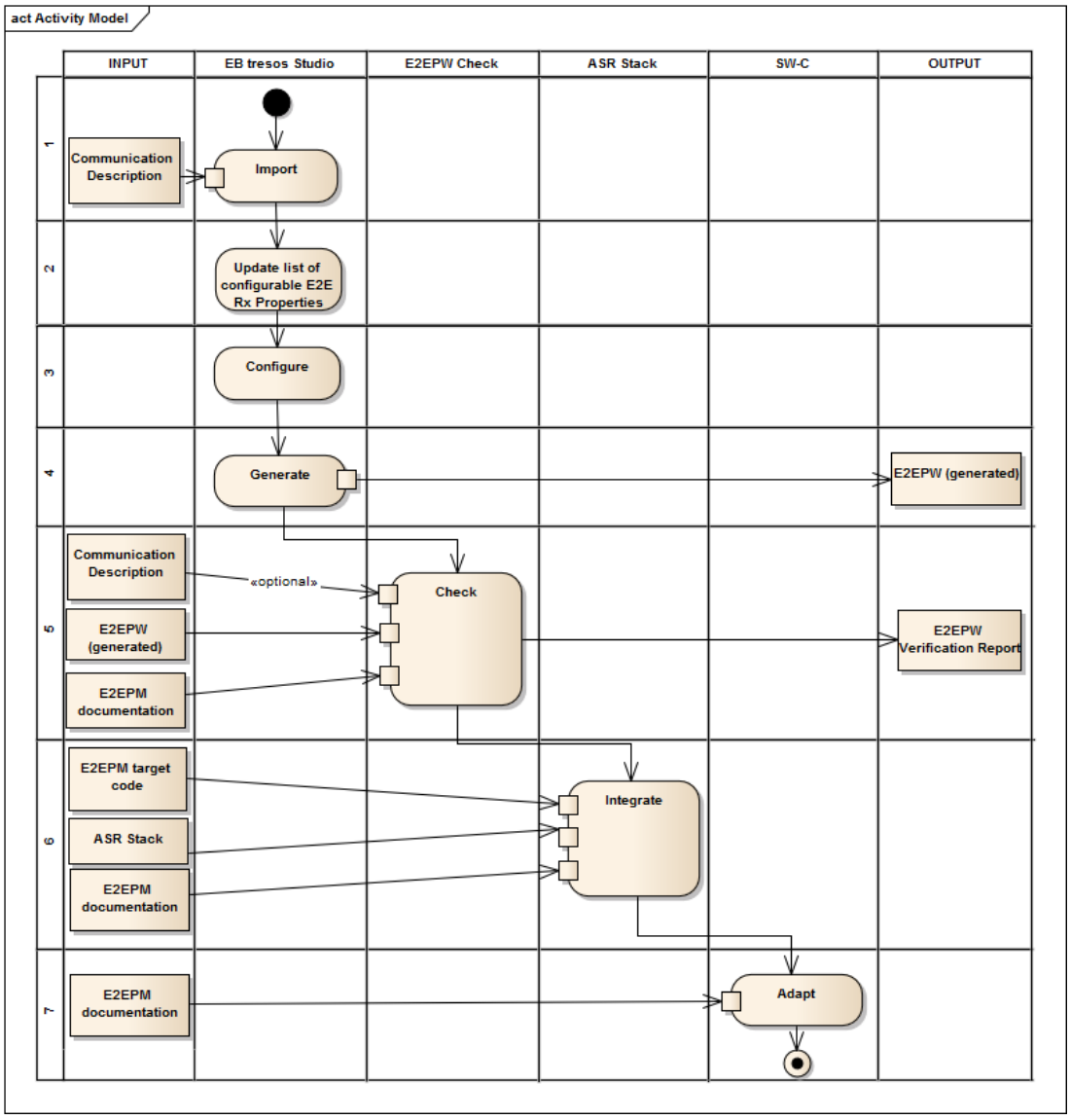

Figure 4.1. E2EPM Workflow

- **Input "Communication Description":** The Communication Description is an ECU extract that conforms to the specification of the AUTOSAR System and Software Component Template. Note that splittable elements can be distributed on multiple physical files. The Communication Description consists of the following content:
  - One instance of the entity SystemTemplate::System.
  - Instances of the entity CoreTopology::EcuInstance.
  - Mapping of Software Component Prototypes onto concrete ECUs using the SwcToEcuMapping class.
  - Mapping of VariableDataPrototype with complex data types to SystemSignals (SenderReceiverToSignalGroupMapping).
  - E2E Protection Information.

- **Workflow step "Import":** A new EB tresos Studio project for an ECU is created which contains at least the modules SCrc, E2EPW, E2E, E2EPXX (XX stands for the delivered profiles, e.g., 02). The Communication Description is imported via the Importer/Exporter dialog.
- **Workflow step "Update list of configurable E2E Rx properties":** The Wizard for updating the list of configurable E2E Rx properties can be called optionally for updating / filling the E2EPW Config with predefined values based on the imported input data. This step simplifies the next workflow step in case additional information must be configured (e.g. maxDeltaCounterInit per receiver).
- **Workflow step "Configure":** Generator specific configuration items (e.g. Wrapper Mode, usage of RteIsUpdated, etc.) are entered in the E2EPW module configuration window. This updates the artefact E2EPW Config.
- **Workflow step "Generate":** The E2EPW MCG generates header and source files containing the E2E Protection wrapper routines for each SW-C based on the configuration.
- **Workflow step "Check":** The generated files are checked with respect to the given input. This process is triggered manually by executing the checker program E2EPWCheck.exe.
- **Workflow step "Integrate":** If the checker report is successful, then the generated files can be integrated into the AUTOSAR communication stack.
- **Workflow step "Adapt":** The application code of the SW-Cs is adapted for the usage with the E2E wrappers (i.e. read and write accesses to the Rte are done via the E2E Protection wrappers).

4.6. Implementing your system

Describe how to implement the whole system using your EB tresos Safety E2E Wrapper. This can consist of:

- Avoiding symbols reserved by EB tresos Safety E2E Wrapper: The user shall not overwrite EB tresos Safety E2E Wrapper symbols with his symbols
- Calling EB tresos Safety E2E Wrapper services: Using functions of this [Safety Product] in a safe way
- Startup checks: Describe the transition between "EB tresos Safety E2E Wrapper is not running" to "EB tresos Safety E2E Wrapper is running".
- Shutdown: Describe the transistion between "EB tresos Safety E2E Wrapper is running" to "EB tresos Safety E2E Wrapper is not running".
- Testing safety mechanisms: If your EB tresos Safety E2E Wrapper provides any safety mechanisms, the user might want to test if they are correctly working at runtime/configuration time. Describe how he can achieve this

5. Safety element out of context (SEooC) definition

E2EPM is developed as a Safety Element out of Context (SEooC) according to [ISO26262-10_DIS].

5.1. Functional scope of the SEooC

5.1.1. Provided functionality

The functional scope of the SEooC is defined by the requirements as specified in Section 3.2.3, “Assumed safety requirements of the EB tresos Safety E2E Wrapper”.

WARNING 	**Assumed safety requirements** The requirements listed in Section 3.2.3, “Assumed safety requirements of the EB tresos Safety E2E Wrapper” are not sufficient to fulfill all safety requirements imposed on the software architecture of the entire system. Depending on the system application, additional safety mechanisms may be necessary to guarantee system safety.

5.1.2. Assumed employment

E2EPM assumes that you employ all of its parts according to the specification given in the safety manual.

5.1.3. Assumed external functionality

E2EPM assumes that the following safety mechanisms are used on the ECU to ensure the integrity of safety related functionality:

- Safe API calls of the E2EPM
- Memory protection for spatial freedom from interference, e.g. provided by EB tresos Safety OS in chase the Single channel wrapper is used
- Control flow monitoring, e.g. provided by *EB tresos Safety TimE*

6. Configuration verification criteria

6.1. Configuration Rules and Notes

6.1.1. Usage of E2EPW Check Tool

The user manual of the E2EPW Check tool is provided in Appendix B, “Safety Checker“.

6.1.2. Configuration Rules

6.1.2.1. Workflow step: Configure

In the following the rules for the workflow step "Configure" are specified:

[EB_E2EPW020168]

The user of the E2EPM shall consider the following for selection of the correct configuration of the configuration parameter *RteIsUpdated*:

- If *RteIsUpdated* is configured to *RteIsUpdatedUsed*, the WrapperMode must be configured to SingleChannel, the Rte need to provide the related API, and the user shall ensure that the following issue is no problem for the correct error handling of the SWC: If the Rte function Rte_IsUpdated erroneously returns FALSE instead of TRUE, E2EPM does not call Rte_Read and directly returns with E2EPW_STATUS_NONEWDATA, i.e. another possible communication error (e.g. wrong CRC) is not indicated.
- If *RteIsUpdated* is configured to *RteIsUpdatedUnused*, the user shall ensure that the following issue is no problem for the correct error handling of the SWC: consecutive wrong CRC errors could result from CRC errors or a single CRC error followed by one or more lost messages.

[EB_E2EPW020082]

The receiver specific E2E Description parameters (i.e maxDeltaCounterInit, maxNoNewOrRepeatedData, and syncCounterInit) shall be equal for all protected receiver ports associated to the same E2E Protection and the same software component.
Note: Rational: parameter are defined as macros which use the short name of the E2E Description and therefore multiple definitions are not possible, Fulfilled by the userguide.

[EB_E2EPW020403]

If the user requires "EB_E2ESE000076: Freedom from Interference regarding memory (S)" the user has to configure the WrapperMode to RedundantWrapper.

Note: Fulfilled by the userguide.

6.1.2.2. Workflow step: Generate E2EPW

In the following the rules for the workflow step "Generate E2EPW" are specified:

[EB_E2EPW020361]

All warning and error messages of the Code generation run should be analyzed by the user.

6.1.2.3. Workflow step: Verify generated files

In the following the rules for the workflow step "Verify generated files" are specified:

[EB_E2EPW020137]

The user manual of the E2EPW Check tool is provided in Appendix B, “Safety Checker“:

- Step 1: Execute E2EPWCheck as described in Section 6.1.1, “Usage of E2EPW Check Tool”.

  All relevant files (i.e. *E2EPW (generated)*) shall be specified as arguments

  Note:

  E2EPW (generated) means the set of *.c and *.h files which are generated into the output directory of the Eb tresos Studio project (i.e. workspace\<E2EProject>\output\generated\include*.h and workspace\<E2EProject>\output\generated\src*.c).

  For a more detailed information on the usage of the tool, execute `E2EPWCheck.exe -help`.

- Step 2: Verify that the return value of E2EPWCheck program is 0.

  If the return value of E2EPWCheck is negative, the program execution failed with an error that is not related to the verification rules and therefore also the verification of the generated E2EPM Target SW shall be considered as failed.

  A non-negative return value denotes the number of verification rule errors, and therefore only a return value of 0 denotes successful verification.

  Hint: The return value of the E2EPW Check program can be determined by calling "echo %errorlevel%" at the command prompt directly after invocation of E2EPW Check.

- Step 3: Verify that the Report (XML) generated by E2EPWCheck exists and that *result* is "OK".

  XPath for attribute *result* which shall contain value 'OK': "/report/@result"

- Step 4: Verify that the Report (XML) does not contain any error messages.

XPath for *error*: "/report/errors/error"

An example is shown in Figure B.5, “E2EPW Check detected error.”.

- Step 5: Perform all reviews as specified by the generated review list of the Report (XML).

  XPath for *review*: "/report/reviews/review"

  XPath for Review description: "./@message"

  XPath for Review objects: "./*" as specified in the review description, e.g. UsingRteIsUpdated, WrapperMode.

  An example is shown in Figure B.4, “E2EPW Verification Report review.”.
- Step 6: Verify that the MD5 checksum of each file given in Step 1 is equal to the "md5" attribute of the respective *checked-file* element of the Report.

  XPath for *checked-file*: "/report/checked-files/checked-file"

  XPath for attribute *md5* which contains the MD5 checksum: "./@md5"

  Verify also that the related self check for each file given in Step 1 is OK.

  XPath for attribute *selfcheck* which shall contain value 'OK': "./@selfcheck"

  An example is shown in Figure B.3, “E2EPW Check arguments and file.”.
- Step 7: Execute E2EPWCheck again with identical parameters and verify that the return value is 0.

  Rationale: to protect against sporadic errors
- Step 8: Verify that the Report (XML) generated by the 2nd E2EPWCheck run exists.
- Step 9: Verify that both generated Reports (XML) are identical except for the time stamp "/report/@timestamp".

6.2. Integration Rules and Notes

6.2.1. Integration Rules and Notes

If the verification report of the workflow step "Verify generated file" (see Figure 4.1, “E2EPM Workflow”) is successful, then the generated files can be integrated into the AUTOSAR communication stack. That is, the application code of the SW-Cs has to be adapted (i.e. read and write accesses to the Rte are done via the E2E Protection wrappers).

If a customer build environment is used, then the static files from the module plugins SCrc, E2E, E2EPXX, and E2EPW

- <TresosBase>\plugins\SCrc_TS_TxDx<ModuleVersionNumber>\include*
- <TresosBase>\plugins\SCrc_TS_TxDx<ModuleVersionNumber>\src*
- <TresosBase>\plugins\E2E_TS_TxDx<ModuleVersionNumber>\include*
- <TresosBase>\plugins\E2E_TS_TxDx<ModuleVersionNumber>\src*
- <TresosBase>\plugins\E2EPXX_TS_TxDx<ModuleVersionNumber>\include*
- <TresosBase>\plugins\E2EPXX_TS_TxDx<ModuleVersionNumber>\src*
- <TresosBase>\plugins\E2EPW_TS_TxDx<ModuleVersionNumber>\include*
- <TresosBase>\plugins\E2EPW_TS_TxDx<ModuleVersionNumber>\src*

and the generated files from

- <TresosBase>\<WORKSPACE>\output\generated\include*
- <TresosBase>\<WORKSPACE>\output\generated\src*

must be copied and integrated into the build environment.

In the following the rules for the workflow step "Integrate" are specified:

[EB_E2EPW020136]

The user of the E2EPM module shall test the correct operation of the receive path for each protected Rx Data Element by receiving each Rx Data Element Member with a value different to 0 without error for at least two times in a row.

The user of the E2EPM module shall test the correct operation of the transmit path for each protected Tx Data Element by transmitting each Tx Data Element Member with a value different to 0 without error for at least two times in a row. In addition the correctness of the transmitted Data Element should be verified by an independent receiver with E2E support or by analyzing the data on the bus.

[EB_E2EPW020138]

The user shall provide the following memory abstraction macros (independent for the AUTOSAR environment):

- `E2EPW_[START|STOP]_SEC_CODE`
- `E2EPW_[START|STOP]_SEC_CONST_UNSPECIFIED`
- `E2E_[START|STOP]_SEC_CODE`
- `SCRC_[START|STOP]_SEC_CODE`

The user shall provide the following memory abstraction macros in an AUTOSAR 3.1, 3.2 or compatible environment:

- E2EPW_[START|STOP]_SEC_<SWCShortname>_VAR_UNSPECIFIED
- E2EPW_[START|STOP]_SEC_RC_<SWCShortname>_VAR_UNSPECIFIED (only if redundant channel is used)
- SCRC_[START|STOP]_SEC_CONST_8BIT

The user shall provide the following memory abstraction macros in an AUTOSAR 4.0 or compatible environment:

- E2EPW_[START|STOP]_SEC_<SWCShortname>_VAR_INIT_UNSPECIFIED
- E2EPW_[START|STOP]_SEC_RC_<SWCShortname>_VAR_INIT_UNSPECIFIED (only if redundant channel is used)
- SCRC_[START|STOP]_SEC_CONST_8

[EB_E2EPW020139]

The following compiler abstractions shall be added to Compiler_Cfg.h:

- E2E_CODE
- E2E_APPL_CONST
- E2E_APPL_DATA
- E2EPW_CODE
- E2EPW_APPL_CONST
- E2EPW_APPL_DATA
- E2EPW_VAR
- E2EPW_CONST
- SCRC_CODE
- SCRC_CONST
- SCRC_APPL_DATA

[EB_E2EPW020001]

The following header files used directly or indirectly by the E2EPM shall comply with the requirements for the development of safety-related software for the automotive domain and shall conform to SWS: Std_Types.h [ASRSWSSTDTYPES], Platform_types.h [ASRSWSPTYPES], Compiler.h [ASRSWSCOMPILER]
Note: Fulfilled by the userguide.

[EB_E2EPW020007]

The memory mapping MemMap.h [ASRMEMORYMAPPING] used by E2EPM shall comply with the requirements for the development of safety-related software for the automotive domain.
Note: Fulfilled by the userguide.

[EB_E2EPW020313]

The user shall ensure that no C-identifier (i.e. an object; a function; a tag or a member of a structure, union, or enumeration; a typedef name; a label name; a macro name; or a macro parameter) is defined with a name that starts with "E2E_-", "E2EPW_-"or "SCRC_-" (case insensitive) outside E2EPM.
Note: Fulfilled by the userguide.

6.3. Application Rules and Notes

6.3.1. Application Rules and Notes

Information and requirements for the use of the E2EPM in an item (e.g. usage of E2EPW instead of direct calls to Rte, error-handling in SWC)

6.3.1.1. Usage of the Protection Wrappers

The usage of the E2EPW is distinguished between the sender and receiver side and consists of several steps as documented in Section 3.2.7.1, “Dynamic Behavior”. These steps are re-used hereinafter for both the single channel and redundant channel wrapper code examples. These code examples are provided without warranty of any kind.

6.3.1.1.1. Sender SW-C (Single channel wrapper)

```
/* Step S0: optionally initialize wrapper (if not done during ECU startup) */
(void)E2EPW_WriteInit_<p>_<o>();

/* Step S1: write Data Element */
AppDataEl.speed = U16_V_MAX;
AppDataEl.accel = U8_0;

/* Step S2: call E2EPW Write */
Ret = E2EPW_Write_<p>_<o>(&AppDataEl);

RetRteWrite = extractRetRteWrite(Ret);
RetE2EPW = extractRetE2EPW(Ret);
RetProtect = extractRetProtect(Ret);
RetStatus = extractRetStatus(Ret);
```

```
/* Step S9: perform error handling */
if((((RetRteWrite != RTE_E_OK) ||
     (RetE2EPW != E2E_E_OK)) ||
     (RetProtect != E2E_E_OK)) ||
     (RetStatus != E2E_E_OK))
{
   /* perform error handling */
}
```

6.3.1.1.2. Receiver SW-C (Single channel wrapper)

```
/* Step R0: optionally initialize wrapper (if not done during ECU startup) */
(void)E2EPW_ReadInit_<p>_<o>();

/* Step R1: call E2EPW Read */
Ret = E2EPW_Read_<p>_<o>(&AppDataEl);

RetRteRead = extractRetRteRead(Ret);
RetE2EPW = extractRetE2EPW(Ret);
RetCheck = extractRetCheck(Ret);
RetCommStatus = extractRetCommStatus(Ret);

if (TRUE == isStartUp)
{
   /* check if data are available */
   if((RetRteRead == RTE_E_OK) &&
      (RetE2EPW == E2E_E_OK)) &&
      (RetCheck == E2E_E_OK)) &&
      (RetCommStatus == E2EPW_STATUS_INITIAL))
   {
      /* Startup phase is over, a valid message has been received */
      isStartUp = FALSE;
   }
   else
   {
       /* Step R0: Initialize E2EPM for Port1/DataElement1 as long as no valid data are
        * available */
      (void)E2EPW_ReadInit_<p>_<o>();
   }

   /* check for startup timeout if required */
```

```
}
else
{
   /* Step R9: check timeout and perform error handling */
   /* check Byte 0, Byte 1, and Byte 2 of return value */
   if((RetRteRead != RTE_E_OK) ||
      (RetE2EPW != E2E_E_OK) ||
      (RetCheck != E2E_E_OK))
   {
      /* check for timeout */

      /* perform error handling */
   }
   else
   {
      /* Check communication status */
      if((RetCommStatus != E2EPW_STATUS_OK) &&
         (RetCommStatus != E2EPW_STATUS_OKSOMELOST))
      {
         /* check for timeout */

         /* perform error handling */
      }
      else
      {
         /* use AppDataEl */
         swc_app(&AppDataEl);
      }
   }
}
```

6.3.1.1.3. Sender SW-C (Redundant channel wrapper)

```
/* Step S0 / Step S10: optionally initialize wrapper (if not done during ECU startup) */
(void)E2EPW_WriteInit1_<p>_<o>();
(void)E2EPW_WriteInit2_<p>_<o>();

/* Step S1: write Data Element */
AppDataEl.speed = U16_V_MAX;
AppDataEl.accel = U8_0;
```

```
/* Step S2: call E2EPW Write1 */
Ret1 = E2EPW_Write1_<p>_<o>(&AppDataEl);

Ret1RteWrite = extractRetRteWrite(Ret1);
Ret1E2EPW = extractRetE2EPW(Ret1);
Ret1Protect = extractRetProtect(Ret1);
Ret1Status = extractRetStatus(Ret1);

/* Step S9: perform error handling */
if((((Ret1RteWrite != RTE_E_OK) ||
     (Ret1E2EPW != E2E_E_OK)) ||
     (Ret1Protect != E2E_E_OK)) ||
     (Ret1Status != E2E_E_OK))
{
   /* perform error handling */
}
else
{
   /* Step S11: write values again to data element */
   AppDataEl.speed = U16_V_MAX;
   AppDataEl.accel = U8_0;

   /* Step S12: call E2EPW Write2 */
   Ret2 = E2EPW_Write2_<p>_<o>(&AppDataEl);

   /* Step S19: error handling */
   if(Ret2 != Ret1)
   {
      /* perform error handling */
   }
}
```

6.3.1.1.4. Receiver SW-C (Redundant channel wrapper)

```
/* Step R0 / Step R10: optionally initialize wrapper (if not done during ECU startup) */
(void)E2EPW_ReadInit1_<p>_<o>();
(void)E2EPW_ReadInit2_<p>_<o>();

/* Step R1: call E2EPW Read 1 */
Ret1 = E2EPW_Read1_<p>_<o>(&AppDataEl);

Ret1RteRead = extractRetRteRead(Ret1);
```

```
Ret1E2EPW = extractRetE2EPW(Ret1);
Ret1Check = extractRetCheck(Ret1);
Ret1CommStatus = extractRetCommStatus(Ret1);

if (TRUE == isStartUp)
{
   /* check if data are available */
   if((Ret1RteRead == RTE_E_OK) &&
      (Ret1E2EPW == E2E_E_OK)) &&
      (Ret1Check == E2E_E_OK)) &&
      (Ret1CommStatus == E2EPW_STATUS_INITIAL))
   {
      /* Startup phase is over, a valid message has been received */
      isStartUp = FALSE;
   }
   else
   {
      /* Step R0: Initialize E2EPM for Port1/DataElement1 as long as no valid data are
       * available */
      (void)E2EPW_ReadInit1_<p>_<o>();
      (void)E2EPW_ReadInit2_<p>_<o>();
   }

   /* check for startup timeout if required */

}
else
{
   /* Step R9: check timeout and perform error handling */
   /* check Byte 0, Byte 1, and Byte 2 of return value */
   if((Ret1RteRead != RTE_E_OK) ||
      (Ret1E2EPW != E2E_E_OK) ||
      (Ret1Check != E2E_E_OK))
   {
      /* check for timeout */

      /* perform error handling */
   }
   else
   {
      /* Check communication status */
      if((Ret1CommStatus != E2EPW_STATUS_OK) &&
         (Ret1CommStatus != E2EPW_STATUS_OKSOMELOST))
      {
         /* check for timeout */

         /* perform error handling */
```

```
      }
      else
      {
         /* copy values from data element */
         RedundantAppDataEl = AppDataEl;
         /* Step R11: call E2EPW Read 2 */
         Ret2 = E2EPW_Read2_<p>_<o>(&AppDataEl);

         /* Step R19: check timeout and perform error handling */
         if(Ret1 != Ret2)
         {
            /* check for timeout */

            /* perform error handling */
         }
         else
         {
            /* check for corruption of AppDataEl after CRC has been checked */
            if((AppDataEl.speed != RedundantAppDataEl.speed) ||
                  (AppDataEl.accel != RedundantAppDataEl.accel))
            {
               /* perform error handling */
            }
            else
            {
               /* use AppDataEl/RedundantAppDataEl */
               swc_appRC(&AppDataEl, &RedundantAppDataEl);
            }
         }
      }
   }
}
```

6.3.1.2. Application Rules

In the following the rules for the workflow step "Adapt" are specified The rules with "ASR_-" as prefix are based on ASR SWS [ASRSWSE2E]. Note that these requirements only cover error handling which is required for the operation of E2EPM.

[ASR_E2EPW020288]

Each E2E Protection Wrapper function belonging to the same data element shall not be called concurrently. Functions belong to the same data element if they share the same postfix <SWC>_<p>_<o>.

Note:

[EB_E2EPW020359]

During the invocation of `E2EPW_Read[1|2]` or `E2EPW_Write[1|2]` the data element shall not be changed by the user.

Note:

[ASR_E2EPW020242]

The SW-C implementation files that invoke E2EPM functions shall include E2EPW_<SWC-Type-short name>.h

Note: Fulfilled by the userguide.

[EB_E2EPW020014]

If the start-up code of the ECU complies with the safety level assigned to the E2EPM in the ECU, the E2EPM internal data structures are considered as correctly initialized. Otherwise, the E2EPM internal data structures shall be initialized by calling the corresponding initialization routine (`E2EPW_ReadInit[1|2]` or `E2EPW_WriteInit[1|2]`) before the first call of any `E2EPW_Read[1|2]` or `E2EPW_Write[1|2]` function.

Note: Fulfilled by the userguide.

[EB_E2EPW020032]

If the E2EPM operates in single channel mode the sender shall invoke once the function `E2EPW_Write_<p>_<o>()` in every send cycle.

[EB_E2EPW020033]

If the E2EPM operates in redundant channel mode the sender shall invoke first once the function `E2EPW_Write1_<p>_<o>()` in every send cycle.

[EB_E2EPW020034]

If E2EPM operates in redundant channel mode and if `E2EPW_Write1_<p>_<o>()` does not return an error, the sender shall invoke the function `E2EPW_Write2_<p>_<o>()`.

[EB_E2EPW020035]

If the E2EPM operates in redundant channel mode and if `E2EPW_Write1_<p>_<o>()` returns an error the sender shall invoke both functions `E2EPW_WriteInit1_<p>_<o>()` and `E2EPW_WriteInit2_<p>_<o>()`.

Note: Additional error handling of the sender is not scope of this requirement. Invoking `E2EPW_WriteInit1_<p>_<o>()` and `E2EPW_WriteInit2_<p>_<o>()` ensures only that the redundant internal state variables of the E2E protection have the same value (init values).

[EB_E2EPW020036]

If the E2EPM operates in redundant channel mode and if `E2EPW_Write2_<p>_<o>()` returns an error the sender shall invoke both functions `E2EPW_WriteInit1_<p>_<o>()` and `E2EPW_WriteInit2_<p>_<o>()`.

Note: Additional error handling of the sender is not scope of this requirement. Invoking `E2EPW_WriteInit1_<p>_<o>()` and `E2EPW_WriteInit2_<p>_<o>()` ensures only that the redundant internal state variables of the E2E protection have the same value (init values).

[EB_E2EPW020037]

If the E2EPM operates in redundant channel mode, the sender shall invoke `E2EPW_Write1_<p>_<o>()` after the invocation of the functions `E2EPW_WriteInit1_<p>_<o>()` and `E2EPW_WriteInit2_<p>_<o>()`.

Note: If the sender cycle shall be restarted, `E2EPW_Write1_<p>_<o>()` (followed by `E2EPW_Write2_<p>_<o>()`) can be invoked again in the same cycle, otherwise in the next send cycle.

[EB_E2EPW020121]

If the initialization functions `E2EPW_ReadInit[1|2]_<p>_<o>(void)` are called after `E2EPW_Read1_<p>_<o>()`, then `E2EPW_Read2_<p>_<o>()` shall not be called afterwards. That is, the redundant wrapper read process again starts with `E2EPW_Read1_<p>_<o>()`.
Note: Fulfilled by the userguide.

[EB_E2EPW020167]

If E2EPM operates in redundant channel mode and if `E2EPW_Read1_<p>_<o>()` does not return an error the receiver shall invoke the function `E2EPW_Read2_<p>_<o>()`.

[ASR_E2EPW020235]

The user of the E2EPM shall only use the received data if the E2EPM indicates the integrity of the data.
Note: Fulfilled by the userguide.

[EB_E2EPW020166]

If delayed reception shall be detected then the user of E2EPM shall check for a timeout after any `E2EPW_Read[1|2]` call.
Note: Fulfilled by the userguide.

[ASR_E2EPW020249]

The integrity of the functionality of the E2E Protection Wrapper (for transmitting/converting safety-related data) shall be guaranteed. The functions of the E2E Protection Wrapper are not reentrant, therefore they are not to be called concurrently.
Note: Fulfilled by the userguide.

[EB_E2EPW020404]

In the case the E2EPM indicates a malfunction, the user shall reset the E2EPM by using the related init function.
Note: Fulfilled by the userguide.

Appendix A. Safety Checklist

The safety manual describes all required activities that need to be performed for a safe usage of the E2EPM. In order to show the fulfilment of all theses activities the checklist shown in Figure A.1, "E2EPM Safety Checklist" can be used. In this reference work flow all activities (categorized as either task or check) are listed according to their related phase within the development process. Each activity is associated with the corresponding chapter of the safety manual, where the detailed requirements and explanations are placed. Furthermore activities are categorized in QM and safety, i.e. QM activities are always required; safety activities are additionally required for achieving functional safety.

Phase	#	Type (Task/ Check)	Activity	Reference to Safety Manual chapter	QM	Safety	Applied (yes, no,partly)	Evidence/Justification/Comment
Delivery	1	T	Download, install	4.2.	x			
	2	C	Check delivery with MD5 checksum	4.3.		x		
	3	C	Check delivery version, check that safety statement correponds to used version	4.1.		x		
Field Monitoring	4	C	Check list of published known issues	4.4.	x			
Technical Safety Concept	5	C	Check that assumed E2EPW requirements are sufficient for the use case	3.2.3		x		
	6	C	Check with OEM that selected E2E profile is appropriate for the use case	3.2.4.1		x		
	7	C	Consider E2EPW Limitations	3.2.12	x			
SW Implementation /Integration	8	T	Configure E2EPW	Refer to EB tresos® E2E Wrapper documentation § 3.5	x			
	9	C	Check Configuration Rules	6.1.2	x			
	10	T	Generate	6.1.2	x			
	11	C	Check Generation Rules	6.1.2.2		x		
	12	T	Execute Check	6.1.1		x		
	13	C	Verify generated files	6.1.2.3		x		
	14	T	Integrate	6.2	x			
	15	C	Check Integration Rules	6.2.1		x		
	16	T	Adapt	6.3.1.1	x			
	17	C	Check Application Rules	6.3	x			
System Integration	18	C	Verify that item sw requirements are fullfilled	3.6.2		x		implicitely covered by Implementation/Integration phase checks
	19	C	Verify that item hw requirements are fullfilled	3.6.3		x		
	20	C	Verify that item tool requirements are fullfilled	3.6.4		x		
	21	C	Verify that communication description notes are fullfilled	3.6.5.1	x			
	22	C	Verify that communication description rules are fullfilled	3.6.5.2		x		

Figure A.1. E2EPM Safety Checklist

Appendix B. Safety Checker

B.1. E2EPW Check Usage

Tool name: E2EPW Check

Tool location: <TresosBase>\plugins\E2EPW_TS_TxDx<ModuleVersionNumber>\check\dist\E2EPWCheck.exe

Note: <TresosBase> is the installation path of EB tresos Studio. <ModuleVersionNumber> contains the version number of the E2EPW module (major and minor number only, patch number always equals 0). For example, if delivered E2EPW module has version 2.3.1, then <ModuleVersionNumber> equals "M2I3R0".

The E2EPW Check tool is fully integrated into the EB tresos Studio workflow (see Figure B.1, “E2EPW Check tool execution within GUI mode.”). Thus, the E2EPW Check tool can be called either from the EB tresos Studio GUI or via command line. NOTE: The execution via the EB tresos Studio GUI does not replace the manual command line execution required in Step 7 of EB_E2EPW020137.

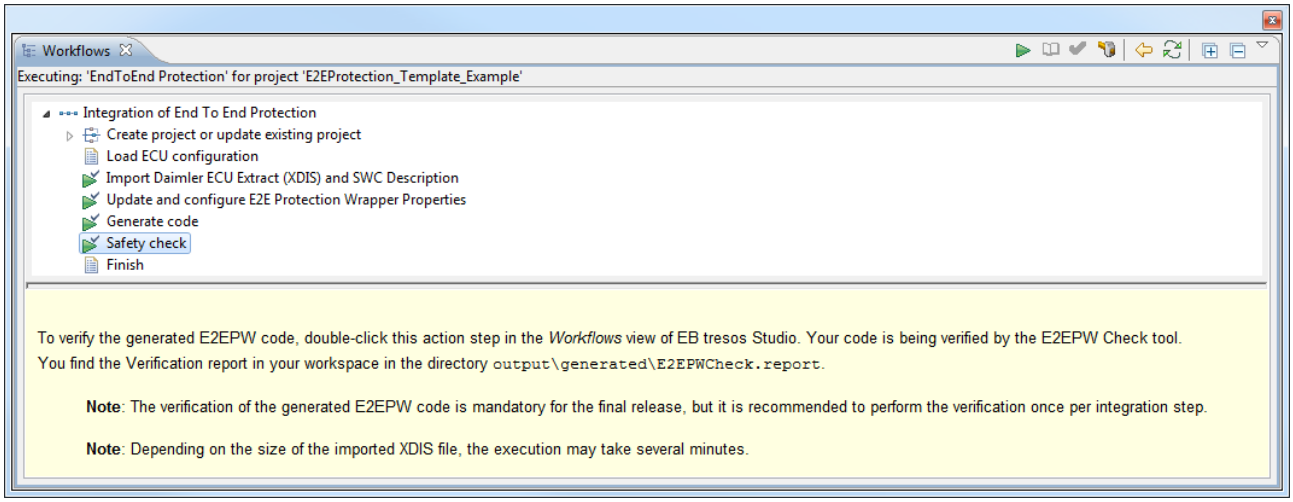

Figure B.1. E2EPW Check tool execution within GUI mode.

The command line arguments of the E2EPW Check tool can be displayed by calling

```
E2EPWCheck.exe --help.
```

During the workflow step "Generate", the E2EPW MCG generates an E2EPW Check configuration file (`E2EPWCheck.cfg`) into the standard project output directory. It contains a list of arguments (e.g. generated E2EPW header and source files) that can be used for execution of E2EPW Check to generate a Verification Report for the specific project.

The call to the E2EPW Check tool by using this configuration file is:

```
E2EPWCheck.exe +E2EPWCheck.cfg
```

with the following exemplary content of `E2EPWCheck.cfg`:

```
--report=C:\EB\Tresos\workspace\ExampleProject\output\generated\E2EPWCheck.report
--ecu-name=TestSender
C:\EB\Tresos\workspace\ExampleProject\output\generated\src\E2EPW_AtomicComponentA.c
C:\EB\Tresos\workspace\ExampleProject\output\generated\include\E2EPW_AtomicComponentA.h
C:\EB\Tresos\workspace\ExampleProject\output\generated\include\E2EPW_Int_Cfg.h
```

B.2. E2EPW Verification Report

E2EPW Check verifies the correct generation of E2EPW (generated) and generates an XML report. The structure of the report is shown in Figure B.2, “E2EPW Verification Report overview.”.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <report executable="path/to/E2EPWChecker" md5="MD5 of E2EPWChecker" result="OK" creator="EB E2EPWCheck version x.y.z"
    timestamp="YYYY/MM/DD hh:mm:ss">
  + <command-line-arguments>
    <input-files />
  + <checked-files>
  + <reviews>
    <errors />
    <warnings />
    <information />
  </report>
```

Figure B.2. E2EPW Verification Report overview.

The attributes of the tag report provide information regarding the creator of the report (Version, name and md5 checksum of executable), the overall result ("FAILED" or "OK") and the generation time of the report (timestamp).

The report sections `<command-line-arguments>`, `<input-files>` and `<checked-files>` document how E2EPW Check has been called. Figure B.3, “E2EPW Check arguments and file.” shows the related tags.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <report executable="path/to/E2EPWChecker" md5="MD5 of E2EPWChecker" result="OK" creator="EB E2EPWCheck version x.y.z"
    timestamp="YYYY/MM/DD hh:mm:ss">
  - <command-line-arguments>
      <command-line-arg name="verbose">False</command-line-arg>
      <command-line-arg name="file">E2EPW_AtomicComponentA.c</command-line-arg>
      <command-line-arg name="file">E2EPW_AtomicComponentA.h</command-line-arg>
      <command-line-arg name="file">E2EPW_Int_Cfg.h</command-line-arg>
      <command-line-arg name="selftest">False</command-line-arg>
      <command-line-arg name="report">E2EPWCheck.report</command-line-arg>
      <command-line-arg name="ecu_name">TestSender</command-line-arg>
    </command-line-arguments>
    <input-files />
  - <checked-files>
      <checked-file md5="024f6839ce47bffda3ab8228ca8b7f1a" selfcheck="OK">E2EPW_AtomicComponentA.c</checked-file>
      <checked-file md5="e1d586f1701a8a17f232f3866d36247c" selfcheck="OK">E2EPW_AtomicComponentA.h</checked-file>
      <checked-file md5="227aa5c7d7a3a524f454664769d57005" selfcheck="OK">E2EPW_Int_Cfg.h</checked-file>
    </checked-files>
  + <reviews>
    <errors />
    <warnings />
    <information />
  </report>
```

Figure B.3. E2EPW Check arguments and file.

The last section of the report (`<information>`) provides information of the E2EPW Check execution and has no relevance for the user.

B.2.1. User Reviews

The report section `<reviews>` specifies the reviews that shall be performed by the user. Figure B.4, “E2EPW Verification Report review.” shows a review example.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <report executable="path/to/E2EPWChecker" md5="MD5 of E2EPWChecker" result="OK" creator="EB E2EPWCheck version x.y.z"
    timestamp="YYYY/MM/DD hh:mm:ss">
  + <command-line-arguments>
    <input-files />
  + <checked-files>
  - <reviews>
      <review message="The usage of the API RteIsUpdated is disabled for all protected receiver ports. Verify that this configuration
        is correct." review-nr="1" id="25" />
      <review message="The E2E Protection Wrapper is generated in single channel mode. Verify that this configuration is correct."
        review-nr="2" id="26" />
    </reviews>
    <errors />
    <warnings />
    <information />
  </report>
```

Figure B.4. E2EPW Verification Report review.

The review rule is specified by the attribute message. Within these rules the terminology that is used within this safety manual applies. Further parameters of the rule are provided as child nodes of the review tag, e.-g. UsingRteIsUpdated or WrapperMode. (The tag with then name rule is for internal testing and tracing and not relevant for the user.)

B.2.2. Detected Errors

The report section `<errors>` specifies the errors detected in the checked files, i.e. the attribute message describes the error and the child nodes provide further information, such as file and line number. (The tag rule is for internal testing and tracing and not relevant for the user.). An example of a detected error is shown in Figure B.5, “E2EPW Check detected error.”.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <report executable="path/to/E2EPWChecker" md5="MD5 of E2EPWChecker" result="FAILED" creator="EB E2EPWCheck version x.y.z"
    timestamp="YYYY/MM/DD hh:mm:ss">
  + <command-line-arguments>
    <input-files />
  + <checked-files>
  + <reviews>
  - <errors>
    - <error message="Macro (E2EPW_CFG_DL_E2EDesc_DE9_P01A) has value (48U), but expected (40U)." id="7">
        <line>645</line>
        <file>E2EPW_AtomicComponentA.c</file>
        <rule>E2EPWVR1005</rule>
      </error>
    </errors>
    <warnings />
    <information />
</report>
```

Figure B.5. E2EPW Check detected error.

Appendix C. Document configuration information

This document has been created by the DocBook engine using the source files and revisions listed below. All paths are relative to the directory https://subversion.ebgroup.elektrobit.com/svn/autosar/asc_E2ESE/asc_-E2ESEmgmt/stable/RFI_ACG-8.8.3-X3_1/doc/safetymanual.

Filename	Revision
../archDesign/E2EPM_autosar.xml	4403
../archDesign/E2EPM_configuration.xml	4403
../archDesign/E2EPM_errordetection.xml	4403
../archDesign/E2EPM_operatingmodes.xml	4403
../archDesign/E2EPM_safetymechanism.xml	4403
../archDesign/E2EPM_sharedressources.xml	4403
../archDesign/swarchdesign.xml	4403
../public/fragments/Bibliography.xml	4403
../public/fragments/Glossary.xml	4403
../requirements/Failure_Model.xml	4403
../requirements/TL-Requirements.xml	4403
Appendix_Safety_Checker.xml	4403
Appendix_Safety_Checklist.xml	4403
ASCE2ESE-16_Safety_E2E_Wrapper_safety_manual.xml	4403
History.xml	4403
SEooC_Definition.xml	4403
SM_About_the_Safety_E2E.xml	4403
SM_Application_Rules.xml	4403
SM_Configuration_Rules.xml	4403
SM_Configuration_verification_criteria.xml	4403
SM_Document_information.xml	4403
SM_Integration_Rules.xml	4403
SM_Using_the_Safety_E2E.xml	4403
userguide/E2EPW_Communication_Description_notes.xml	4403
userguide/E2EPW_Communication_Description_P01A_notes.xml	4403

Glossary

E2EPW Checker Tool	The Safety Transformer Checker is a tool to verify the generated code from the E2EPW. In this way the confidence in the generated code is increased.
Communication Description	The Communication Description holds the information about the network, the communication matrix, and the protection of signals.
E2E	The E2E module is the end-to-end protection library, that includes the protection and the check routines.
E2EPW	The E2EPW calls the *E2E* library to either protect the serialized data stream in the sending path or checks the received data stream for communication errors.
EB tresos Safety E2E Wrapper	The EB tresos Safety E2E Wrapper consists of the following modules: ► *E2EPW* ► *E2E* ► *SCrC* ► E2E profiles
E2EPM	E2E protection module (short E2EPM), refer to *EB tresos safety E2E Wrapper*
Integrator	An integrator is a person who integrates the software on the ECU.
Rte	The Runtime Environment or also AUTOSAR Runtime Environment is an AUTOSAR module.
SCrc	The SCrc module in the safety implementation of the CRC.
Software component (SWC)	A software component (SWC) is an AUTOSAR software component.
EB tresos Studio	EB tresos Studio provides the user with a GUI for administration and controlling of the Target SW components. EB tresos Studio is invoked by the *Integrator* and provides an interface to import the AUTOSAR *Communication Description* into an internal database which can be accessed by EB tresos Studio Public APIs.
System description	A system description is an AUTOSAR system description.
EB treses safety TimE	EB treses safety TimE is safety qualified module for Temporal and Logical Program Flow Monitoring.

Bibliography

[ASRMEMO-RYMAPPING] *AUTOSAR Specification of Memory Mapping, V1.4.0 R4.0 Rev3*

[ASRSWS-COMPILER] *AUTOSAR Specification of Compiler Abstraction, V3.2.0 R4.0 Rev3 and V2.2.0 R3.2 Rev2*

[ASRSWSE2E] *Specification of SW-C End-to-End Communication Protection Library V2.0.0, R4.0 Rev 3 and V2.1.0, R3.2 Rev 2, 2012*, 2012

[ASRSWSPTYPES] *AUTOSAR Specification of Platform Types, V2.5.0 R4.0 Rev3 and V2.3.1 R3.2 Rev1*

[ASRSWSST-DTYPES] *AUTOSAR Specification of Standard Types, V1.3.0 R4.0 Rev1 and V1.5.0 R3.2 Rev2*

[ASRSWSSWCT] *AUTOSAR Specification of Software Component Template, V4.2.0, R3*

[EBASCE2ESE--24] *E2EPM Release Notes, Safety_E2E_Wrapper_release_notes.pdf*,

[EBASCE2ESE--33] *E2EPM Integration Test Specification*

[ISO26262_1ST] *INTERNATIONAL STANDARD ISO 26262*: Road vehicles - Functional safety, 2011
http://wiki.elektrobit.com/index.php/ISO26262

[ISO26262-10_DIS] *INTERNATIONAL STANDARD ISO 26262-10*: Road vehicles - Functional safety - Part 10: Guideline on ISO 26262 analyses, 2011
http://qms/portal/1.0.1/release/other_content/guidances/examples/resources/ISO_26262-10_(E)_2011-01.pdf