Document Title	Specification of SW-C End-to-End Communication Protection Library
Document Category	Software specification (SWS)
Document UID	428
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History

Date	Release	Changed by	Description
2026-11-26	R26-11	AUTOSAR Release Management	• Removed references to RS Libraries • Removed 'Extended Production Errors' chapter • Aligned chapters with CP SWS Template • Removed the <Mip>_GetVersionInfo API from chapter 8 and also reference in other parts of the document to this functionality • In SWS_E2E_91068, replaced init value
2025-11-27	R25-11	AUTOSAR Release Management	• Add 2 new functions for E2E for methods • Remove use cases • Remove all references to E2EPW
2024-11-27	R24-11	AUTOSAR Release Management	• E2EPW Support removed • Profile 76 added • Change Interfaces of Profile 22
2023-11-23	R23-11	AUTOSAR Release Management	• Corrections of Length type in P44m
2022-11-24	R22-11	AUTOSAR Release Management	• Corrections of MinDataLength in P04m and P07m • Correction of syntax errors in profiles P11 and P22 • Chapters 8 and Annex B: Make Service IDs of functions unique



△

2021-11-25	R21-11	AUTOSAR Release Management	• New profiles 8m, 44m
2020-11-30	R20-11	AUTOSAR Release Management	• New profiles 4m, 7m, 08,44 • E2E for methods • Extension of E2E State Machine
2019-11-28	R19-11	AUTOSAR Release Management	• Incorporated E2E_PxxForward methods to replicate detected E2E-Errors on outgoing messages • E2E P0xSTATUS_ERROR values are now the same for all profiles • Fixed minor inconsistencies and typos • Updated Tracing from SRS_E2E to RS_E2E • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Added clarification regarding assumptions on failure modes and detection capabilities in annex A. • Fixed inconsistent definition of length in E2E header for P04, P05, and P06 • Clarification of parameters CounterOffset and CRCOffset in E2E_P01ConfigType
2017-12-08	4.3.1	AUTOSAR Release Management	• Updated traceability to SRS E2E. • Fixed enumeration literals for E2E_PxxCheckStatusType for profiles 1 and 2. • Corrected name of step E2E_SMClearProfileStatus to E2E_SMClearStatus in Routine E2E_SM_checkinit • Various clarifications in configuration and routine parameters, mainly of profile 2 and 7.

▽

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Added new Profiles 7, 11 and 22. • Fixed initialization of profile 1 and 2 in the init function. Now properly sets WaitForFirstData to TRUE. • Corrected/unified initialization of Counter state variable and bit/byte conversion in configuration data in profiles 4, 5, and 6. • Removed chapter 8.3.7 elementary protocol functions that were marked obsolete since several releases.
2015-07-31	4.2.2	AUTOSAR Release Management	• Introduced new E2E state machine profile status E2E_P_NONEWDATA. Adapted figures, API tables and mapping functions. This solves an issue with deterministic startup of the state machine. • Updated Figure 7 7, added behavior in case ReceivedCounter is out of range. • Assigned new specification ID SWS_E2E_00478 to duplicate specification SWS_E2E_00324 (specification of profile 4). • Fixed "Calculate CRC over Data ID and Data", which was already fixed in R4.1.2 but falsely included as of R4.1.1.
2014-10-31	4.2.1	AUTOSAR Release Management	• Introduction of E2E profiles 4, 5, 6 • Introduction of E2E state machine • Introduction of init functions and status mapping functions for profiles 1, 2 • Overview of wrapper, by means of several new diagrams.
2014-03-31	4.1.3	AUTOSAR Release Management	• Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Correction in E2E variant 1C • Various minor corrections • Editorial changes

▽

△

2013-03-15	4.1.1	AUTOSAR Release Management	• Full support for E2E protection at signal group level • Removed dependency to Rte_IsUpdated • Changed recommendations about the maximum data lengths • Addition of initialization functions to the redundant wrapper • Corrections in code examples • Reworked according to the new SWS_BSWGeneral • New indexing scheme for requirements • Extension of E2E Profile 1 to support 12-bit Data IDs (variant 1C) • Alignment with ISO 26262 (terms, communication faults) • Quality ameliorations (due to document review) • Clarification in the configuration of E2E parameters
2011-12-22	4.0.3	AUTOSAR Release Management	• E2E Profile 3 removed (not backward compatible) • Several bugfixes in of E2E Protection Wrapper API (not backward compatible) • Modified return values of E2E Protection Wrapper API (not backward compatible) • Addition of init API for the E2E Protection Wrapper • Several bugfixes and modifications in code examples of E2E Protection Wrapper • Extensions in configuration, making sender and receiver more independent • Bugfix in the profile 1 alternating mode CRC calculation • Clarifications with in E2E Profile 1 with respect to the CRC • Several minor bug fixes • Several optimizations in the text descriptions

▽

△

			△ • New template with requirements traceability
2010-09-30	3.1.5	AUTOSAR Release Management	• Corrected the wrapper configuration. • Corrected the code example for the usage of the wrapper.
2010-02-02	3.1.4	AUTOSAR Release Management	• Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 14
2 Acronyms and Abbreviations 16
3 Related documentation 17
3.1 Input documents & related standards and norms . . . 17
3.2 Related specification . . . 17
4 Constraints and assumptions 18
4.1 Limitations . . . 18
4.2 Implementation of the E2E Library . . . 18
5 Dependencies to other modules 19
5.1 Required file structure . . . 19
5.2 Dependency on CRC library . . . 19
6 Requirements Tracing 21
7 Functional specification 26
7.1 Error classification . . . 26
7.1.1 Development Errors . . . 26
7.1.2 Runtime Errors . . . 27
7.1.3 Production Errors . . . 27
8 API specification 28
8.1 Imported types . . . 28
8.2 Type definitions . . . 28
8.2.1 E2E Profile 1 types . . . 29
8.2.1.1 E2E_P01ConfigType . . . 29
8.2.1.2 E2E_P01DataIDMode . . . 30
8.2.1.3 E2E_P01ProtectStateType . . . 31
8.2.1.4 E2E_P01CheckStateType . . . 32
8.2.1.5 E2E_P01CheckStatusType . . . 33
8.2.2 E2E Profile 2 types . . . 34
8.2.2.1 E2E_P02ConfigType . . . 35
8.2.2.2 E2E_P02ProtectStateType . . . 36
8.2.2.3 E2E_P02CheckStateType . . . 36
8.2.2.4 E2E_P02CheckStatusType . . . 37
8.2.3 E2E Profile 4 types . . . 38
8.2.3.1 E2E_P04ConfigType . . . 39
8.2.3.2 E2E_P04ProtectStateType . . . 40
8.2.3.3 E2E_P04CheckStateType . . . 40
8.2.3.4 E2E_P04CheckStatusType . . . 41
8.2.3.5 E2E_P04HeaderInformationType . . . 42
8.2.4 E2E Profile 4m types . . . 42

8.2.4.1 E2E_P04mConfigType . . . 42
8.2.4.2 E2E_P04mProtectStateType . . . 43
8.2.4.3 E2E_P04mCheckStateType . . . 44
8.2.4.4 E2E_P04mCheckStatusType . . . 44
8.2.4.5 E2E_P04mHeaderInformationType . . . 45
8.2.5 E2E Profile 5 types . . . 46
8.2.5.1 E2E_P05ConfigType . . . 46
8.2.5.2 E2E_P05ProtectStateType . . . 47
8.2.5.3 E2E_P05CheckStateType . . . 47
8.2.5.4 E2E_P05CheckStatusType . . . 48
8.2.5.5 E2E_P05HeaderInformationType . . . 49
8.2.6 E2E Profile 6 types . . . 49
8.2.6.1 E2E_P06ConfigType . . . 49
8.2.6.2 E2E_P06ProtectStateType . . . 50
8.2.6.3 E2E_P06CheckStateType . . . 51
8.2.6.4 E2E_P06CheckStatusType . . . 52
8.2.6.5 E2E_P06HeaderInformationType . . . 52
8.2.7 E2E Profile 7 types . . . 53
8.2.7.1 E2E_P07ConfigType . . . 53
8.2.7.2 E2E_P07ProtectStateType . . . 54
8.2.7.3 E2E_P07CheckStateType . . . 54
8.2.7.4 E2E_P07CheckStatusType . . . 55
8.2.7.5 E2E_P07HeaderInformationType . . . 56
8.2.8 E2E Profile 7m types . . . 56
8.2.8.1 E2E_P07mConfigType . . . 56
8.2.8.2 E2E_P07mProtectStateType . . . 57
8.2.8.3 E2E_P07mCheckStateType . . . 58
8.2.8.4 E2E_P07mCheckStatusType . . . 59
8.2.8.5 E2E_P07mHeaderInformationType . . . 59
8.2.9 E2E Profile 8 types . . . 60
8.2.9.1 E2E_P08ConfigType . . . 60
8.2.9.2 E2E_P08ProtectStateType . . . 61
8.2.9.3 E2E_P08CheckStateType . . . 61
8.2.9.4 E2E_P08CheckStatusType . . . 62
8.2.9.5 E2E_P08HeaderInformationType . . . 63
8.2.10 E2E Profile 8m types . . . 64
8.2.10.1 E2E_P08mConfigType . . . 64
8.2.10.2 E2E_P08mProtectStateType . . . 65
8.2.10.3 E2E_P08mCheckStateType . . . 65
8.2.10.4 E2E_P08mCheckStatusType . . . 66
8.2.10.5 E2E_P08mHeaderInformationType . . . 67
8.2.11 E2E Profile 11 types . . . 67
8.2.11.1 E2E_P11ConfigType . . . 67

8.2.11.2 E2E_P11DataIDMode . . . 69
8.2.11.3 E2E_P11ProtectStateType . . . 69
8.2.11.4 E2E_P11CheckStateType . . . 70
8.2.11.5 E2E_P11CheckStatusType . . . 70
8.2.12 E2E Profile 22 types . . . 71
8.2.12.1 E2E_P22ConfigType . . . 71
8.2.12.2 E2E_P22ProtectStateType . . . 72
8.2.12.3 E2E_P22CheckStateType . . . 73
8.2.12.4 E2E_P22CheckStatusType . . . 74
8.2.13 E2E Profile 44 types . . . 75
8.2.13.1 E2E_P44ConfigType . . . 75
8.2.13.2 E2E_P44ProtectStateType . . . 76
8.2.13.3 E2E_P44CheckStateType . . . 76
8.2.13.4 E2E_P44CheckStatusType . . . 77
8.2.13.5 E2E_P44HeaderInformationType . . . 77
8.2.14 E2E Profile 44m types . . . 78
8.2.14.1 E2E_P44mConfigType . . . 78
8.2.14.2 E2E_ P44mProtectStateType . . . 79
8.2.14.3 E2E_P44mCheckStateType . . . 79
8.2.14.4 E2E_P44mCheckStatusType . . . 80
8.2.14.5 E2E_P44mHeaderInformationType . . . 81
8.2.15 E2E Profile 76 Types . . . 82
8.2.15.1 E2E_P76ConfigType . . . 82
8.2.15.2 E2E_P76ProtectStateType . . . 83
8.2.15.3 E2E_P76CheckStateType . . . 83
8.2.15.4 E2E_P76CheckStatusType . . . 84
8.2.16 E2E state machine types . . . 85
8.2.16.1 E2E_PCheckStatusType . . . 85
8.2.16.2 E2E_SMConfigType . . . 85
8.2.16.3 E2E_SMCheckStateType . . . 86
8.2.16.4 E2E_SMStateType . . . 88
8.3 Function definitions . . . 88
8.3.1 E2E Profile 1 routines . . . 89
8.3.1.1 E2E_P01Protect . . . 89
8.3.1.2 E2E_P01ProtectInit . . . 89
8.3.1.3 E2E_P01Forward . . . 90
8.3.1.4 E2E_P01Check . . . 91
8.3.1.5 E2E_P01CheckInit . . . 91
8.3.1.6 E2E_P01MapStatusToSM . . . 92
8.3.2 E2E Profile 2 routines . . . 94
8.3.2.1 E2E_P02Protect . . . 94
8.3.2.2 E2E_P02ProtectInit . . . 95
8.3.2.3 E2E_P02Forward . . . 96

8.3.2.4 E2E_P02Check . . . 96
8.3.2.5 E2E_P02CheckInit . . . 97
8.3.2.6 E2E_P02MapStatusToSM . . . 98
8.3.3 E2E Profile 4 routines . . . 100
8.3.3.1 E2E_P04Protect . . . 100
8.3.3.2 E2E_P04ProtectInit . . . 100
8.3.3.3 E2E_P04Forward . . . 101
8.3.3.4 E2E_P04Check . . . 102
8.3.3.5 E2E_P04CheckInit . . . 102
8.3.3.6 E2E_P04MapStatusToSM . . . 103
8.3.3.7 E2E_P04GetHeaderInfo . . . 104
8.3.4 E2E Profile 4m routines . . . 105
8.3.4.1 E2E_P04mProtect . . . 105
8.3.4.2 E2E_P04mProtectInit . . . 105
8.3.4.3 E2E_P04mForward . . . 106
8.3.4.4 E2E_P04mSourceCheck . . . 107
8.3.4.5 E2E_P04mSinkCheck . . . 108
8.3.4.6 E2E_P04mCheckInit . . . 109
8.3.4.7 E2E_P04mMapStatusToSM . . . 109
8.3.4.8 E2E_P04mGetHeaderInfo . . . 111
8.3.5 E2E Profile 5 routines . . . 111
8.3.5.1 E2E_P05Protect . . . 111
8.3.5.2 E2E_P05ProtectInit . . . 112
8.3.5.3 E2E_P05Forward . . . 113
8.3.5.4 E2E_P05Check . . . 113
8.3.5.5 E2E_P05CheckInit . . . 114
8.3.5.6 E2E_P05MapStatusToSM . . . 115
8.3.5.7 E2E_P05GetHeaderInfo . . . 116
8.3.6 E2E Profile 6 routines . . . 117
8.3.6.1 E2E_P06Protect . . . 117
8.3.6.2 E2E_P06ProtectInit . . . 117
8.3.6.3 E2E_P06Forward . . . 118
8.3.6.4 E2E_P06Check . . . 119
8.3.6.5 E2E_P06CheckInit . . . 119
8.3.6.6 E2E_P06MapStatusToSM . . . 120
8.3.6.7 E2E_P06GetHeaderInfo . . . 121
8.3.7 E2E Profile 7 routines . . . 122
8.3.7.1 E2E_P07Protect . . . 122
8.3.7.2 E2E_P07ProtectInit . . . 122
8.3.7.3 E2E_P07Forward . . . 123
8.3.7.4 E2E_P07Check . . . 124
8.3.7.5 E2E_P07CheckInit . . . 125
8.3.7.6 E2E_P07MapStatusToSM . . . 125

8.3.7.7 E2E_P07GetHeaderInfo . . . 127
8.3.8 E2E Profile 7m routines . . . 127
8.3.8.1 E2E_P07mProtect . . . 127
8.3.8.2 E2E_P07mProtectInit . . . 128
8.3.8.3 E2E_P07mForward . . . 129
8.3.8.4 E2E_P07mSourceCheck . . . 130
8.3.8.5 E2E_P07mSinkCheck . . . 131
8.3.8.6 E2E_P07mCheckInit . . . 132
8.3.8.7 E2E_P07mMapStatusToSM . . . 132
8.3.8.8 E2E_P07mGetHeaderInfo . . . 134
8.3.9 E2E Profile 8 routines . . . 134
8.3.9.1 E2E_P08Protect . . . 134
8.3.9.2 E2E_P08ProtectInit . . . 135
8.3.9.3 E2E_P08Forward . . . 136
8.3.9.4 E2E_P08Check . . . 136
8.3.9.5 E2E_P08CheckInit . . . 137
8.3.9.6 E2E_P08MapStatusToSM . . . 138
8.3.9.7 E2E_P08GetHeaderInfo . . . 139
8.3.10 E2E Profile 8m routines . . . 140
8.3.10.1 E2E_P08mProtect . . . 140
8.3.10.2 E2E_P08mProtectInit . . . 140
8.3.10.3 E2E_P08mForward . . . 141
8.3.10.4 E2E_P08mSourceCheck . . . 142
8.3.10.5 E2E_P08mSinkCheck . . . 143
8.3.10.6 E2E_P08mCheckInit . . . 144
8.3.10.7 E2E_P08mMapStatusToSM . . . 144
8.3.10.8 E2E_P08mGetHeaderInfo . . . 146
8.3.11 E2E Profile 11 routines . . . 146
8.3.11.1 E2E_P11Protect . . . 146
8.3.11.2 E2E_P11ProtectInit . . . 147
8.3.11.3 E2E_P11Forward . . . 148
8.3.11.4 E2E_P11Check . . . 148
8.3.11.5 E2E_P11CheckInit . . . 149
8.3.11.6 E2E_P11MapStatusToSM . . . 150
8.3.12 E2E Profile 22 routines . . . 151
8.3.12.1 E2E_P22Protect . . . 151
8.3.12.2 E2E_P22ProtectInit . . . 152
8.3.12.3 E2E_P22Forward . . . 152
8.3.12.4 E2E_P22Check . . . 153
8.3.12.5 E2E_P22CheckInit . . . 154
8.3.12.6 E2E_P22MapStatusToSM . . . 154
8.3.13 E2E Profile 44 routines . . . 156
8.3.13.1 E2E_P44Protect . . . 156

8.3.13.2 E2E_P44ProtectInit . . . 156
8.3.13.3 E2E_P44Forward . . . 157
8.3.13.4 E2E_P44Check . . . 158
8.3.13.5 E2E_P44CheckInit . . . 158
8.3.13.6 E2E_P44MapStatusToSM . . . 159
8.3.13.7 E2E_P44GetHeaderInfo . . . 160
8.3.14 E2E Profile 44m routines . . . 161
8.3.14.1 E2E_P44mProtect . . . 161
8.3.14.2 E2E_P44mProtectInit . . . 161
8.3.14.3 E2E_P44mForward . . . 162
8.3.14.4 E2E_P44mSourceCheck . . . 163
8.3.14.5 E2E_P44mSinkCheck . . . 164
8.3.14.6 E2E_P44mCheckInit . . . 165
8.3.14.7 E2E_P44mMapStatusToSM . . . 165
8.3.14.8 E2E_P44mGetHeaderInfo . . . 167
8.3.15 E2E Profile 76 routines . . . 167
8.3.15.1 E2E_P76Protect . . . 167
8.3.15.2 E2E_P76ProtectInit . . . 168
8.3.15.3 E2E_P76Check . . . 169
8.3.15.4 E2E_P76CheckInit . . . 170
8.3.15.5 E2E_P76MapStatusToSM . . . 171
8.3.16 E2E State machine routines . . . 172
8.3.16.1 E2E_SMCheck . . . 172
8.3.16.2 E2E_SMCheckInit . . . 173
8.4 Scheduled functions . . . 173
8.5 Expected interfaces . . . 174
8.5.1 Mandatory Interfaces . . . 174
9 Sequence diagrams 175
9.1 Sender . . . 175
9.1.1 Sender of data elements . . . 175
9.1.2 Sender at signal group level . . . 177
9.2 Receiver . . . 178
9.2.1 Receiver at data element level . . . 181
9.2.2 Receiver at signal group level . . . 183
10 Configuration specification 185
10.1 Published Information . . . 185
A Annex A: Safety Manual for usage of E2E Library 186
A.1 E2E profiles and their standard variants . . . 186
A.2 E2E error handling . . . 186
A.3 Methodology of usage of E2E Library . . . 186
A.4 RTE configuration constraints for SW-C level protection . . . 187
A.4.1 Communication model for SW-C level protection . . . 187

A.4.2 Multiplicities for SW-C level protection . . . 187
A.4.3 Explicit access . . . 188
A.5 Restrictions on the use of COM features . . . 188
B Annex B: Application hints on usage of E2E Library 191
B.1 COM E2E Callouts . . . 192
B.1.1 Functional overview . . . 192
B.1.1.1 Sending/Calling . . . 194
B.1.1.2 Reception . . . 195
B.1.2 Methodology . . . 196
B.1.3 Code Example . . . 197
B.2 Protection at RTE level through E2E Transformer . . . 198
C Not applicable requirements 199
D Change history of AUTOSAR traceable items 200
D.1 Traceable item history of this document according to AUTOSAR Release R22-11 . . . 200
D.1.1 Added Specification Items in R22-11 . . . 200
D.1.2 Changed Specification Items in R22-11 . . . 201
D.1.3 Deleted Specification Items in R22-11 . . . 201
D.2 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 202
D.2.1 Added Specification Items in R23-11 . . . 202
D.2.2 Changed Specification Items in R23-11 . . . 202
D.2.3 Deleted Specification Items in R23-11 . . . 202
D.3 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 202
D.3.1 Added Specification Items in R24-11 . . . 202
D.3.2 Changed Specification Items in R24-11 . . . 202
D.3.3 Deleted Specification Items in R24-11 . . . 203
D.4 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 203
D.4.1 Added Specification Items in R25-11 . . . 203
D.4.2 Changed Specification Items in R25-11 . . . 203
D.4.3 Deleted Specification Items in R25-11 . . . 204
D.5 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 204
D.5.1 Added Specification Items in R26-11 . . . 204
D.5.2 Changed Specification Items in R26-11 . . . 204
D.5.3 Deleted Specification Items in R26-11 . . . 204

1 Introduction and functional overview

This document cointains the platform specific implementation requirements of the PRS E2E Protocol. This includes interfaces and datatypes used.

The main part of the functional specification is given in the AUTOSAR Foundation document 849 "E2E Protocol Specification".

Platform dependent functional specifications extending the protocol specifications are collected in the following sub section(s).

The concept of E2E protection assumes that safety-related data exchange shall be protected at runtime against the effects of faults within the communication link (see Figure 1.1). Examples for such faults are random HW faults (e.g. corrupt registers of a CAN transceiver), interference (e.g. due to EMC), and systematic faults within the software implementing the VFB communication (e.g. RTE, IOC, COM and network stacks).

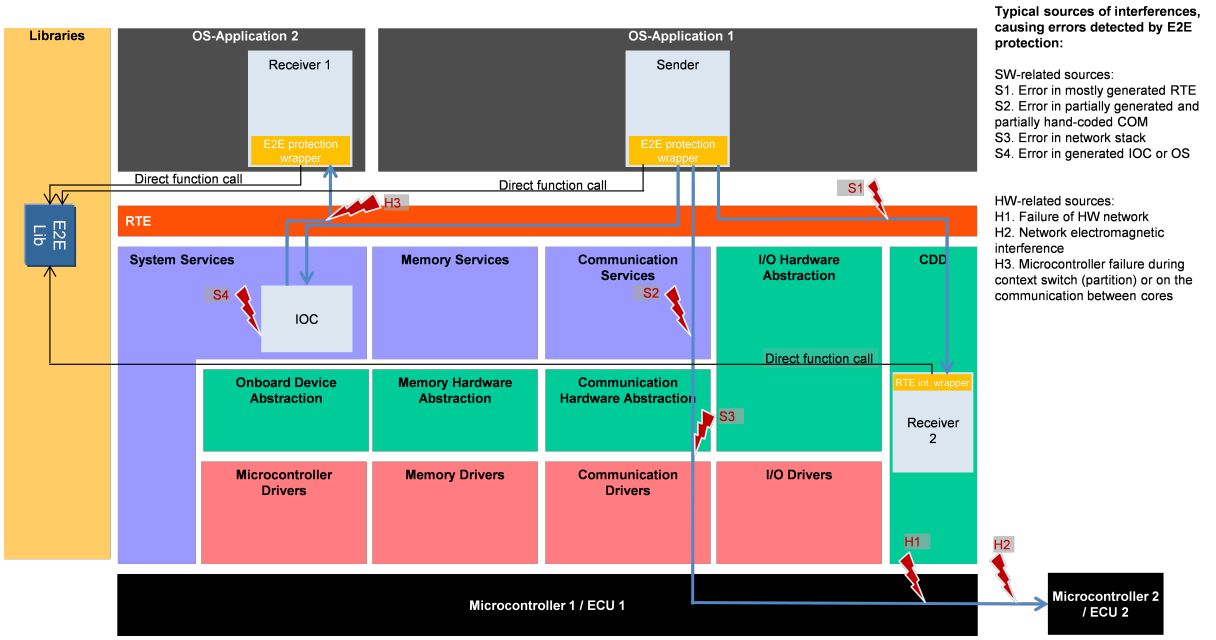


Figure 1.1: Example of faults mitigated by E2E protection

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

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

The E2E protectionallows the following:

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

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

The E2E library is invoked from:

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

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

- In case E2E Transformer is used, the serialization is performed by a transformer above E2E Transformer (COM-based transformer or Some/IP transformer).
- In case the COM callout is used, the serialization is done by the communication stack (RTE, COM), so the callout operates directly on the serialized signal groups in the I-PDU.

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

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

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

2 Acronyms and Abbreviations

All technical terms used in this document, except the ones listed in the table below, can be found in the official AUTOSAR glossary [1, AUTOSAR_TR_Glossary].

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

Abbreviation / Acronym:	Description:
E2E Library	Short name for the End-to-End Communication Protection Library.
Data ID	An identifier that uniquely identifies the message / data element / data.
Source ID	An identifier that uniquely identified the source (origin) of the message / data element / data.
Repetition	Repetition of information.
Loss	Loss of information.
Delay	Delay of information.
Insertion	Insertion of information.
Masquerade	Masquerade.
Incorrect addressing	Incorrect addressing of information.
Incorrect sequence	Incorrect sequence of information.
Corruption	Corruption of information.
Asymmetric information	Asymmetric information sent from a sender to multiple receivers.
Subset	Information from a sender received by only a subset of the receivers.
Blocking	Blocking access to a communication channel.

Table 2.1: Acronyms and abbreviations used in the scope of this Document

In the whole document, there are many requirements that apply to all E2E Profiles at the same time.Such requirements are defined as one requirement that applies to all profiles at the same time. In case some names are profile dependent, then XX notation is used: if in a requirement appears the string containing XX, then it is developed to strings with 01, 02, 04, 4m, 05, 06, 07, 7m, 11, 12, 22, and 44 respectively instead of XX. For example, E2E_PXXCheck() develops to the following two E2E_P01Check(), E2E_P02Check() a.s.o.

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] E2E Protocol Specification
AUTOSAR_FO_PRS_E2EProtocol

[4] Requirements on E2E
AUTOSAR_FO_RS_E2E

[5] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[6] System Template
AUTOSAR_CP_TPS_SystemTemplate

[7] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[8] Specification of ECU Configuration
AUTOSAR_CP_TPS_ECUConfiguration

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for E2ELibrary.

Thus, the specification SWS BSW General shall be considered as additional and required specification for E2ELibrary.

4 Constraints and assumptions

4.1 Limitations

General limitations regarding E2E protection and the detectable failure modes are described in [3, AUTOSAR PRS E2EProtocol]. E2E Profile 2 has in R4.2.1 a new setting offset. This offset can be configured in the system template. However, the E2E Profile 2 specification does not support the case when offset is different than 0. The specification of E2E Profile 2 will be fixed in a future AUTOSAR release, to support a configurable offset.

E2E Profile 1 in the "Double Data ID configuration"uses an implicit 2-byte Data ID, over which CRC8 is calculated. As a CRC over two different 2-byte numbers may result with the same CRC, some precautions must be taken by the user. See PRS_E2E_UC_00072 and PRS_E2E_UC_00073.

E2E Profile 2 uses an implicit 1-byte Data ID, selected from a List of Data IDs depending on each value of the counter, for calculation of the CRC. See [3, AUTOSAR PRS E2EProtocol] for details on the usage and generation of DataIDList for E2E profile 2.

4.2 Implementation of the E2E Library

[SWS_E2E_00050]

Upstream requirements: RS_E2E_08527

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

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

[SWS_E2E_00311]

Upstream requirements: RS_E2E_08528

⌈The configuration of the E2E Library and of the code invoking it (e.g. E2E wrapper , E2E callouts , E2E transformer) shall be implemented and configured (including configuration options used from other subsystems, e.g. COM signal to I-PDU mapping) according to the requirements for the development of safety-related software for the automotive domain.⌋

5 Dependencies to other modules

5.1 Required file structure

[SWS_E2E_00048]

Upstream requirements: RS_E2E_08528

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

[SWS_E2E_00215]

Upstream requirements: RS_E2E_08528

⌈Files E2E_PXX.c and E2E.h shall contain implementation parts specific of each profile.⌋

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

[SWS_E2E_00115]

Upstream requirements: RS_E2E_08528

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

5.2 Dependency on CRC library

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

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

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

To have the same results of the functions E2E_P01Protect() and E2E_P01Check() in >=R4.0 and R3.2, while using differently functioning CRC library, E2E "compensates" different behavior of the CRC library. This results with different invocation of the CRC library by E2E library in >=R4.0 and R3.2.

Mandatory interfaces to CRC Library are listed here:

[SWS_E2E_91095] Definition of mandatory interfaces required by module E2E

Upstream requirements: RS_E2E_08530, RS_E2E_08533, RS_E2E_08531

⌈

API Function	*Header File*	*Description*
Crc_CalculateCRC16	Crc.h	This service makes a CRC16 calculation on Crc_Length data bytes.
Crc_CalculateCRC32P4	Crc.h	This service makes a CRC32 calculation on Crc_Length data bytes, using the polynomial 0xF4ACFB13. This CRC routine is used by E2E Profile 4.
Crc_CalculateCRC64	Crc.h	This service makes a CRC64 calculation on Crc_Length data bytes, using the polynomial 0x42F0E1EBA9EA3693. This CRC routine is used by E2E Profile 7.
Crc_CalculateCRC8	Crc.h	This service makes a CRC8 calculation on Crc_Length data bytes, with SAE J1850 parameters
Crc_CalculateCRC8H2F	Crc.h	This service makes a CRC8 calculation with the Polynomial 0x2F on Crc_Length

⌋

6 Requirements Tracing

The following tables reference the requirements specified in [4, Requirements on E2E], [5, General Requirements on Basic Software Modules] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_E2E_08527]	Implementation of E2E protocol shall fulfill ISO 26262	[SWS_E2E_00050] [SWS_E2E_00158] [SWS_E2E_00160] [SWS_E2E_00161] [SWS_E2E_00166] [SWS_E2E_00338] [SWS_E2E_00339] [SWS_E2E_00349] [SWS_E2E_00350] [SWS_E2E_00373] [SWS_E2E_00379] [SWS_E2E_00382] [SWS_E2E_00385] [SWS_E2E_00387] [SWS_E2E_00390] [SWS_E2E_00391] [SWS_E2E_00393] [SWS_E2E_00446] [SWS_E2E_00447] [SWS_E2E_00449] [SWS_E2E_00450] [SWS_E2E_00452] [SWS_E2E_00455] [SWS_E2E_00457] [SWS_E2E_00458] [SWS_E2E_00460] [SWS_E2E_00546] [SWS_E2E_00547] [SWS_E2E_00548] [SWS_E2E_00549] [SWS_E2E_00550] [SWS_E2E_00572] [SWS_E2E_00573] [SWS_E2E_00574] [SWS_E2E_00575] [SWS_E2E_00576] [SWS_E2E_00577] [SWS_E2E_00578] [SWS_E2E_00579] [SWS_E2E_00580] [SWS_E2E_00581] [SWS_E2E_00583] [SWS_E2E_00584] [SWS_E2E_00585] [SWS_E2E_00586] [SWS_E2E_00587] [SWS_E2E_00588] [SWS_E2E_00589] [SWS_E2E_00590] [SWS_E2E_00614] [SWS_E2E_00615] [SWS_E2E_91001] [SWS_E2E_91002] [SWS_E2E_91003] [SWS_E2E_91004] [SWS_E2E_91005] [SWS_E2E_91006] [SWS_E2E_91007] [SWS_E2E_91012] [SWS_E2E_91013] [SWS_E2E_91014] [SWS_E2E_91015] [SWS_E2E_91016] [SWS_E2E_91017] [SWS_E2E_91018] [SWS_E2E_91027] [SWS_E2E_91028] [SWS_E2E_91029] [SWS_E2E_91030] [SWS_E2E_91031] [SWS_E2E_91032] [SWS_E2E_91033] [SWS_E2E_91034] [SWS_E2E_91035] [SWS_E2E_91036] [SWS_E2E_91037] [SWS_E2E_91038] [SWS_E2E_91039] [SWS_E2E_91040] [SWS_E2E_91041] [SWS_E2E_91059] [SWS_E2E_91070] [SWS_E2E_91071] [SWS_E2E_91073] [SWS_E2E_91074] [SWS_E2E_91075] [SWS_E2E_91076] [SWS_E2E_91077] [SWS_E2E_91078] [SWS_E2E_91079] [SWS_E2E_91080] [SWS_E2E_91081] [SWS_E2E_91082] [SWS_E2E_91083] [SWS_E2E_91084] [SWS_E2E_91085] [SWS_E2E_91086] [SWS_E2E_91087] [SWS_E2E_91088] [SWS_E2E_91089] [SWS_E2E_91090] [SWS_E2E_91091] [SWS_E2E_91092] [SWS_E2E_91093] [SWS_E2E_91094] [SWS_E2E_91098] [SWS_E2E_91099] [SWS_E2E_91100] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_E2E_91101] [SWS_E2E_91102] [SWS_E2E_91113] [SWS_E2E_91114] [SWS_E2E_91115] [SWS_E2E_91117] [SWS_E2E_91118] [SWS_E2E_91119] [SWS_E2E_91120] [SWS_E2E_91121] [SWS_E2E_91122]
[RS_E2E_08528]	E2E protocol shall provide different E2E profiles	[SWS_E2E_00011] [SWS_E2E_00017] [SWS_E2E_00018] [SWS_E2E_00020] [SWS_E2E_00021] [SWS_E2E_00048] [SWS_E2E_00110] [SWS_E2E_00115] [SWS_E2E_00152] [SWS_E2E_00153] [SWS_E2E_00154] [SWS_E2E_00158] [SWS_E2E_00160] [SWS_E2E_00161] [SWS_E2E_00166] [SWS_E2E_00200] [SWS_E2E_00215] [SWS_E2E_00311] [SWS_E2E_00334] [SWS_E2E_00335] [SWS_E2E_00336] [SWS_E2E_00337] [SWS_E2E_00338] [SWS_E2E_00339] [SWS_E2E_00349] [SWS_E2E_00350] [SWS_E2E_00353] [SWS_E2E_00370] [SWS_E2E_00371] [SWS_E2E_00373] [SWS_E2E_00378] [SWS_E2E_00379] [SWS_E2E_00380] [SWS_E2E_00381] [SWS_E2E_00382] [SWS_E2E_00383] [SWS_E2E_00384] [SWS_E2E_00386] [SWS_E2E_00387] [SWS_E2E_00388] [SWS_E2E_00389] [SWS_E2E_00390] [SWS_E2E_00391] [SWS_E2E_00392] [SWS_E2E_00393] [SWS_E2E_00437] [SWS_E2E_00438] [SWS_E2E_00439] [SWS_E2E_00440] [SWS_E2E_00441] [SWS_E2E_00443] [SWS_E2E_00444] [SWS_E2E_00445] [SWS_E2E_00446] [SWS_E2E_00447] [SWS_E2E_00449] [SWS_E2E_00450] [SWS_E2E_00451] [SWS_E2E_00452] [SWS_E2E_00455] [SWS_E2E_00456] [SWS_E2E_00457] [SWS_E2E_00458] [SWS_E2E_00459] [SWS_E2E_00460] [SWS_E2E_00476] [SWS_E2E_00477] [SWS_E2E_00542] [SWS_E2E_00544] [SWS_E2E_00545] [SWS_E2E_00546] [SWS_E2E_00547] [SWS_E2E_00548] [SWS_E2E_00549] [SWS_E2E_00550] [SWS_E2E_00551] [SWS_E2E_00552] [SWS_E2E_00555] [SWS_E2E_00556] [SWS_E2E_00557] [SWS_E2E_00558] [SWS_E2E_00559] [SWS_E2E_00560] [SWS_E2E_00561] [SWS_E2E_00562] [SWS_E2E_00563] [SWS_E2E_00564] [SWS_E2E_00565] [SWS_E2E_00566] [SWS_E2E_00567] [SWS_E2E_00568] [SWS_E2E_00569] [SWS_E2E_00570] [SWS_E2E_00571] [SWS_E2E_00572] [SWS_E2E_00573] [SWS_E2E_00574] [SWS_E2E_00575] [SWS_E2E_00576] [SWS_E2E_00577] [SWS_E2E_00578] [SWS_E2E_00579] [SWS_E2E_00580] [SWS_E2E_00581] [SWS_E2E_00583] [SWS_E2E_00584] [SWS_E2E_00585] [SWS_E2E_00586] [SWS_E2E_00587] [SWS_E2E_00588] [SWS_E2E_00589] [SWS_E2E_00590] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_E2E_00591] [SWS_E2E_00601] [SWS_E2E_00602] [SWS_E2E_00605] [SWS_E2E_00606] [SWS_E2E_00608] [SWS_E2E_00609] [SWS_E2E_00614] [SWS_E2E_00615] [SWS_E2E_00616] [SWS_E2E_00617] [SWS_E2E_00618] [SWS_E2E_00619] [SWS_E2E_00624] [SWS_E2E_00625] [SWS_E2E_10002] [SWS_E2E_10004] [SWS_E2E_10005] [SWS_E2E_91021] [SWS_E2E_91023] [SWS_E2E_91024] [SWS_E2E_91025] [SWS_E2E_91026] [SWS_E2E_91027] [SWS_E2E_91028] [SWS_E2E_91042] [SWS_E2E_91053] [SWS_E2E_91059] [SWS_E2E_91060] [SWS_E2E_91063] [SWS_E2E_91068] [SWS_E2E_91070] [SWS_E2E_91071] [SWS_E2E_91072] [SWS_E2E_91073] [SWS_E2E_91074] [SWS_E2E_91075] [SWS_E2E_91076] [SWS_E2E_91077] [SWS_E2E_91078] [SWS_E2E_91079] [SWS_E2E_91080] [SWS_E2E_91081] [SWS_E2E_91082] [SWS_E2E_91083] [SWS_E2E_91084] [SWS_E2E_91085] [SWS_E2E_91086] [SWS_E2E_91087] [SWS_E2E_91088] [SWS_E2E_91089] [SWS_E2E_91090] [SWS_E2E_91091] [SWS_E2E_91092] [SWS_E2E_91093] [SWS_E2E_91094] [SWS_E2E_91103] [SWS_E2E_91104] [SWS_E2E_91105] [SWS_E2E_91106] [SWS_E2E_91107] [SWS_E2E_91108] [SWS_E2E_91109] [SWS_E2E_91110] [SWS_E2E_91111] [SWS_E2E_91112] [SWS_E2E_91113] [SWS_E2E_91114] [SWS_E2E_91115] [SWS_E2E_91117] [SWS_E2E_91118] [SWS_E2E_91119] [SWS_E2E_91120] [SWS_E2E_91121] [SWS_E2E_91122] [UC_E2E_00053] [UC_E2E_00202] [UC_E2E_00203] [UC_E2E_00204] [UC_E2E_00205] [UC_E2E_00206] [UC_E2E_00207] [UC_E2E_00209] [UC_E2E_00230] [UC_E2E_00232] [UC_E2E_00233] [UC_E2E_00235] [UC_E2E_00250] [UC_E2E_00251] [UC_E2E_00258] [UC_E2E_00270] [UC_E2E_00271] [UC_E2E_00277] [UC_E2E_00278] [UC_E2E_00290] [UC_E2E_00313]
[RS_E2E_08530]	Each E2E profile shall define a set of protection mechanisms and its behavior	[SWS_E2E_91095]
[RS_E2E_08531]	E2E Library shall call the CRC routines of CRC library	[SWS_E2E_91095]
[RS_E2E_08533]	CRC used in a E2E profile shall be different than the CRC used by the underlying physical communication protocol	[SWS_E2E_91095]

▽

△

Requirement	Description	Satisfied by
[RS_E2E_08534]	E2E protocol shall provide E2E Check status to the application	[SWS_E2E_00021] [SWS_E2E_00022] [SWS_E2E_00047] [SWS_E2E_00049] [SWS_E2E_00154] [SWS_E2E_00214] [SWS_E2E_00216] [SWS_E2E_00336] [SWS_E2E_00337] [SWS_E2E_00439] [SWS_E2E_00440] [SWS_E2E_00444] [SWS_E2E_00445] [SWS_E2E_00542] [SWS_E2E_00563] [SWS_E2E_00564] [SWS_E2E_00568] [SWS_E2E_00569] [SWS_E2E_00591] [SWS_E2E_00617] [SWS_E2E_00618] [SWS_E2E_91008] [SWS_E2E_91009] [SWS_E2E_91019] [SWS_E2E_91022] [SWS_E2E_91025] [SWS_E2E_91026] [SWS_E2E_91035] [SWS_E2E_91075] [SWS_E2E_91076] [SWS_E2E_91079] [SWS_E2E_91080] [SWS_E2E_91103] [SWS_E2E_91104] [SWS_E2E_91105] [SWS_E2E_91106] [SWS_E2E_91107] [SWS_E2E_91108] [SWS_E2E_91109] [SWS_E2E_91110] [SWS_E2E_91111] [SWS_E2E_91112]
[RS_E2E_08538]	An E2E Transformer shall be provided	[SWS_E2E_00037]
[RS_E2E_08539]	An E2E protection mechanism for inter-ECU communication of short to large data shall be provided	[SWS_E2E_00334] [SWS_E2E_00335] [SWS_E2E_00336] [SWS_E2E_00338] [SWS_E2E_00339] [SWS_E2E_00349] [SWS_E2E_00350] [SWS_E2E_00373] [SWS_E2E_00377] [SWS_E2E_00378] [SWS_E2E_00385] [SWS_E2E_00393] [SWS_E2E_00437] [SWS_E2E_00438] [SWS_E2E_00439] [SWS_E2E_00440] [SWS_E2E_00441] [SWS_E2E_00443] [SWS_E2E_00444] [SWS_E2E_00445] [SWS_E2E_00446] [SWS_E2E_00447] [SWS_E2E_00448] [SWS_E2E_00449] [SWS_E2E_00450] [SWS_E2E_00451] [SWS_E2E_00452] [SWS_E2E_00453] [SWS_E2E_00455] [SWS_E2E_00456] [SWS_E2E_00457] [SWS_E2E_00458] [SWS_E2E_00459] [SWS_E2E_00460] [SWS_E2E_00461] [SWS_E2E_00542] [SWS_E2E_00544] [SWS_E2E_00545] [SWS_E2E_00546] [SWS_E2E_00547] [SWS_E2E_00548] [SWS_E2E_00549] [SWS_E2E_00550] [SWS_E2E_00551] [SWS_E2E_00552] [SWS_E2E_00584] [SWS_E2E_00585] [SWS_E2E_00586] [SWS_E2E_00590] [SWS_E2E_00592] [SWS_E2E_00593] [SWS_E2E_00594] [SWS_E2E_00595] [SWS_E2E_00596] [SWS_E2E_00597] [SWS_E2E_00598] [SWS_E2E_00599] [SWS_E2E_00603] [SWS_E2E_00604] [SWS_E2E_00610] [SWS_E2E_00611] [SWS_E2E_00614] [SWS_E2E_00615] [SWS_E2E_00616] [SWS_E2E_00617] [SWS_E2E_00619] [SWS_E2E_00624] [SWS_E2E_00625] [SWS_E2E_10001] [SWS_E2E_10002] [SWS_E2E_10004] [SWS_E2E_10005] [SWS_E2E_91001] [SWS_E2E_91002] [SWS_E2E_91003] [SWS_E2E_91004] [SWS_E2E_91005] [SWS_E2E_91006] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_E2E_91007] [SWS_E2E_91008] [SWS_E2E_91010] [SWS_E2E_91011] [SWS_E2E_91012] [SWS_E2E_91013] [SWS_E2E_91014] [SWS_E2E_91015] [SWS_E2E_91016] [SWS_E2E_91017] [SWS_E2E_91018] [SWS_E2E_91019] [SWS_E2E_91020] [SWS_E2E_91036] [SWS_E2E_91037] [SWS_E2E_91038] [SWS_E2E_91039] [SWS_E2E_91040] [SWS_E2E_91041] [SWS_E2E_91042] [SWS_E2E_91059] [SWS_E2E_91070] [SWS_E2E_91098] [SWS_E2E_91099] [SWS_E2E_91100] [SWS_E2E_91101] [SWS_E2E_91102]
[RS_E2E_08548]	E2E protocol shall provide E2E overall state to the application	[SWS_E2E_00340] [SWS_E2E_00342] [SWS_E2E_00343] [SWS_E2E_00344] [SWS_E2E_00347] [SWS_E2E_00351] [SWS_E2E_00352] [SWS_E2E_00380] [SWS_E2E_00381] [SWS_E2E_00383] [SWS_E2E_00384] [SWS_E2E_00453] [SWS_E2E_00454] [SWS_E2E_00461] [SWS_E2E_00462] [SWS_E2E_00476] [SWS_E2E_00477] [SWS_E2E_00553] [SWS_E2E_00554] [SWS_E2E_00557] [SWS_E2E_00558] [SWS_E2E_00561] [SWS_E2E_00562] [SWS_E2E_00600] [SWS_E2E_00601] [SWS_E2E_00602] [SWS_E2E_00603] [SWS_E2E_00604] [SWS_E2E_00605] [SWS_E2E_00606] [SWS_E2E_00607] [SWS_E2E_00608] [SWS_E2E_00609] [SWS_E2E_00612] [SWS_E2E_00613] [SWS_E2E_00623] [SWS_E2E_00626] [SWS_E2E_00627] [SWS_E2E_10003] [SWS_E2E_10006] [SWS_E2E_10007] [SWS_E2E_91060]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_E2E_00047]
[SRS_BSW_00337]	Classification of development errors	[SWS_E2E_00047]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_E2E_00038]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Error classification

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

[SWS_E2E_00049]

Upstream requirements: RS_E2E_08534

⌈The E2E library shall not contain library-internal mechanisms for error detection to be traced as development errors.⌋

[SWS_E2E_00011]

Upstream requirements: RS_E2E_08528

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

[SWS_E2E_00216]

Upstream requirements: RS_E2E_08534

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

7.1.1 Development Errors

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

[SWS_E2E_00047] Definition of development errors in module E2E

Upstream requirements: SRS_BSW_00337, SRS_BSW_00323, RS_E2E_08534

⌈

Type of error	*Related error code*	*Error value*
At least one pointer parameter is a NULL pointer	E2E_E_INPUTERR_NULL	0x13
At least one input parameter is erroneous, e.g. out of range	E2E_E_INPUTERR_WRONG	0x17
Function executed in wrong state	E2E_E_WRONGSTATE	0x1A

⌋

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

[UC_E2E_00313]

Upstream requirements: RS_E2E_08528

⌈The caller of the E2E functions E2E_PXXProtect() / E2E_PXXCheck() shall handle the errors/stati defined in SWS_E2E_00047 according to the column "How do caller of E2E shall handle it".⌋

In other words, the E2E libary does not define any integration errors for itself. However, the caller of E2E library uses the return values of E2E functions and does the corresponding error handling.

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Production Errors

There are no production errors.

8 API specification

This chapter specifies the API of E2E Library.

Members of the configuration structures (e.g. in Figure 8.1) are in alphabetical order. However, for implementation, the sequence of members of this data structure is provided by table specification items (e.g. [SWS_E2E_00386]).

8.1 Imported types

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

[SWS_E2E_00017] Definition of imported datatypes of module E2E

Upstream requirements: RS_E2E_08528

⌈

Module	*Header File*	*Imported Type*
Std	Std_Types.h	Std_MessageResultType
	Std_Types.h	Std_MessageTypeType
	Std_Types.h	Std_ReturnType

⌋

8.2 Type definitions

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

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

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

Example 1 - Counter with bit offset = 8 on MSB microcontroller:

	MSB							LSB
Data[0]	7	6	5	4	3	2	1	0
	CRC with bit offset 0							
Data[1]	15	14	13	12	11	10	9	8
	User data with bit offset 12				Counter with offset 8			
Data[2]	23	22	21	20	19	18	17	16
	User data with bit offset 20				User data with bit offset 16			

8.2.1 E2E Profile 1 types

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

typedef E2E_P01ProtectStateType E2E_P01SenderStateType;

typedef E2E_P01CheckStateType E2E_P01ReceiverStateType;

typedef E2E_P01CheckStatusType E2E_P01ReceiverStatusType;

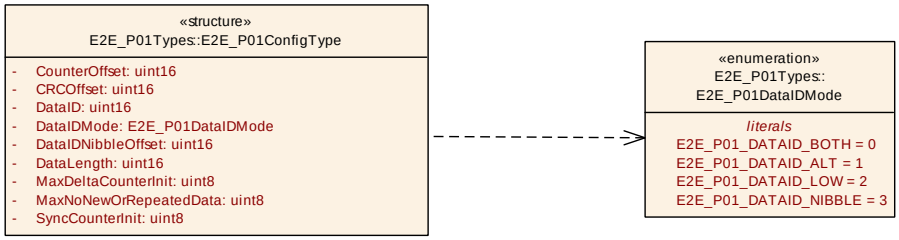


Figure 8.1: E2E Profile 1 configuration

8.2.1.1 E2E_P01ConfigType

[SWS_E2E_00018] Definition of datatype E2E_P01ConfigType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P01ConfigType	
Kind	Structure	
Elements	CounterOffset	
	Type	uint16
	Comment	Bit offset of Counter in MSB first order. CounterOffset shall be a multiple of 4. In variants 1A, 1B, and 1C, CounterOffset is 8.
	CRCOffset	
	Type	uint16
	Comment	Bit offset of CRC (i.e. since *Data) in MSB first order. The offset shall be a multiple of 8. In variants 1A, 1B, and 1C, CRCOffset is 0.
	DataID	
	Type	uint16
	Comment	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section "Configuration constraints on Data IDs".
	DataIDNibbleOffset	
	Type	uint16

▽

△

	Comment	Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P01_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than E2E_P01_DATAID_NIBBLE, Data IDNibbleOffset shall be initialized to 0 (even if it is ignored by E2E Library).
	DataIDMode	
	Type	E2E_P01DataIDMode
	Comment	Inclusion mode of ID in CRC computation (both bytes, alternating, or low byte only of ID included).
	DataLength	
	Type	uint16
	Comment	Length of data, in bits. The value shall be a multiple of 8.
	MaxDeltaCounterInit	
	Type	uint8
	Comment	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	MaxNoNewOrRepeatedData	
	Type	uint8
	Comment	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	SyncCounterInit	
	Type	uint8
	Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 1. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.1.2 E2E_P01DataIDMode

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

[SWS_E2E_00200] Definition of datatype E2E_P01DataIDMode

Upstream requirements: RS_E2E_08528

⌈

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

⌋

8.2.1.3 E2E_P01ProtectStateType

[SWS_E2E_00020] Definition of datatype E2E_P01ProtectStateType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P01ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that the first Data will have the counter 0. After the protection by the Counter, the Counter is incremented modulo 0xF. The value 0xF is skipped (after 0xE the next is 0x0), as 0xF value represents the error value. The four high bits are always 0.
Description	State of the sender for a Data protected with E2E Profile 1.	
Available via	E2E.h	

⌋

8.2.1.4 E2E_P01CheckStateType

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

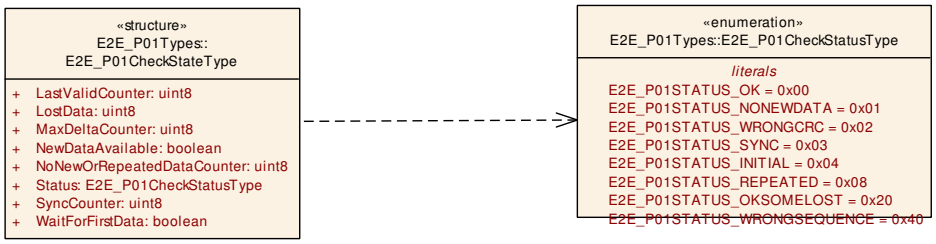


Figure 8.2: E2E Profile 1 check state type

[SWS_E2E_00021] Definition of datatype E2E_P01CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P01CheckStateType	
Kind	Structure	
Elements	LastValidCounter	
	Type	uint8
	Comment	Counter value most recently received. If no data has been yet received, then the value is 0x0. After each reception, the counter is updated with the value received.
	MaxDeltaCounter	
	Type	uint8
	Comment	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
	WaitForFirstData	
	Type	boolean
	Comment	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	NewDataAvailable	
	Type	boolean
	Comment	Indicates to E2E Library that a new data is available for Library to be checked. This attribute is set by the E2E Library caller, and not by the E2E Library.
	LostData	
	Type	uint8
	Comment	Number of data (messages) lost since reception of last valid one. This attribute is set only if Status equals E2E_P01STATUS_OK or E2E_P01 STATUS_OKSOMELOST. For other values of Status, the value of Lost Data is undefined. E2E_P01CheckStatusType Status Result of the verification of the Data, determined by the Check function.
	Status	
	Type	E2E_P01CheckStatusType

▽

△

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

⌋

8.2.1.5 E2E_P01CheckStatusType

[SWS_E2E_00022] Definition of datatype E2E_P01CheckStatusType

Upstream requirements: RS_E2E_08534

⌈

Name	E2E_P01CheckStatusType		
Kind	Enumeration		
Range	E2E_P01STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P01STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P01STATUS_WRONGCRC	0x02	Error: The data has been received according to communication medium, but 1. the CRC is incorrect (applicable for all E2E Profile 1 configurations) or 2. the low nibble of the high byte of Data ID is incorrect (applicable only for E2E Profile 1 with E2E_P01DataIDMode = E2E_P01_DATAID_NIBBLE). The two above errors can be a result of corruption, incorrect addressing or masquerade.

▽

△

	E2E_P01STATUS_SYNC	0x03	NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
	E2E_P01STATUS_INITIAL	0x04	Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.
	E2E_P01STATUS_ REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _ INITIAL, _OK, or _OKSOMELOST.
	E2E_P01STATUS_ OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = Max DeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P01STATUS_ WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description	Result of the verification of the Data in E2E Profile 1, determined by the Check function.		
Available via	E2E.h		

⌋

8.2.2 E2E Profile 2 types

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

typedef E2E_P02ProtectStateType E2E_P02SenderStateType;

typedef E2E_P02CheckStateType E2E_P02ReceiverStateType;

typedef E2E_P02CheckStatusType E2E_P02ReceiverStatusType;

8.2.2.1 E2E_P02ConfigType

[SWS_E2E_00152] Definition of datatype E2E_P02ConfigType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P02ConfigType	
Kind	Structure	
Elements	DataLength	
	Type	uint16
	Comment	Length of Data, in bits. The value shall be a multiple of 8.
	DataIDList	
	Type	Array of uint8
	Size	16
	Comment	An array of appropriately chosen Data IDs for protection against masquerading.
	MaxDeltaCounterInit	
	Type	uint8
	Comment	Initial maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.
	MaxNoNewOrRepeatedData	
	Type	uint8
	Comment	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
	SyncCounterInit	
	Type	uint8
	Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
	Offset	
	Type	uint16
	Comment	Offset of the E2E header in the Data[] array in bits. It shall be: 0 <= Offset <= DataLength-(2*8).
Description	Non-modifiable configuration of the data element sent over an RTE port, for E2E profile 2. The position of the counter and CRC is not configurable in profile 2.	
Available via	E2E.h	

⌋

8.2.2.2 E2E_P02ProtectStateType

[SWS_E2E_00153] Definition of datatype E2E_P02ProtectStateType

Upstream requirements: RS_E2E_08528

⌈

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

⌋

8.2.2.3 E2E_P02CheckStateType

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

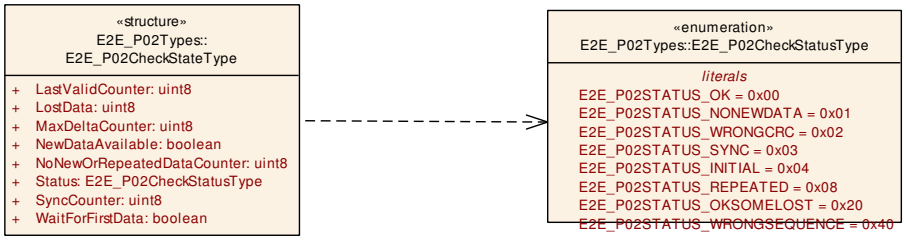


Figure 8.3: E2E Profile 2 check state

[SWS_E2E_00154] Definition of datatype E2E_P02CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P02CheckStateType	
Kind	Structure	
Elements	LastValidCounter	
	Type	uint8
	Comment	Counter of last valid received message.
	MaxDeltaCounter	
	Type	uint8
	Comment	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
	WaitForFirstData	
	Type	boolean

▽

△

	Comment	If true means that no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
	NewDataAvailable	
	Type	boolean
	Comment	Indicates to E2E Library that a new data is available for Library to be checked. This attribute is set by the E2E Library caller, and not by the E2E Library.
	LostData	
	Type	uint8
	Comment	Number of data (messages) lost since reception of last valid one.
	Status	
	Type	E2E_P02CheckStatusType
	Comment	Result of the verification of the Data, determined by the Check function.
	SyncCounter	
	Type	uint8
	Comment	Number of Data required for validating the consistency of the counter that must be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.
	NoNewOrRepeatedDataCounter	
	Type	uint8
	Comment	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.
Description	State of the sender for a Data protected with E2E Profile 2.	
Available via	E2E.h	

⌋

8.2.2.4 E2E_P02CheckStatusType

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

[SWS_E2E_00214] Definition of datatype E2E_P02CheckStatusType

Upstream requirements: RS_E2E_08534

⌈

Name	E2E_P02CheckStatusType		
Kind	Enumeration		
Range	E2E_P02STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _ OKSOMELOST. This means that no Data has been lost since the last correct data reception.

▽

△

	E2E_P02STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P02STATUS_ WRONGCRC	0x02	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P02STATUS_SYNC	0x03	NOT VALID: The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
	E2E_P02STATUS_INITIAL	0x04	Initial: The new data has been received according to communication medium, the CRC is correct, but this is the first Data since the receiver's initialization or reinitialization, so the Counter cannot be verified yet.
	E2E_P02STATUS_ REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _ INITIAL, _OK, or _OKSOMELOST.
	E2E_P02STATUS_ OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter =Max DeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P02STATUS_ WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description	Result of the verification of the Data in E2E Profile 2, determined by the Check function.		
Available via	E2E.h		

⌋

8.2.3 E2E Profile 4 types

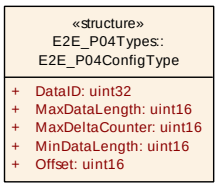


Figure 8.4: E2E Profile 4 configuration

8.2.3.1 E2E_P04ConfigType

[SWS_E2E_00334] Definition of datatype E2E_P04ConfigType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P04ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint16
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(12*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint16
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00651]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint16
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00651]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint16
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 4. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.3.2 E2E_P04ProtectStateType

[SWS_E2E_00335] Definition of datatype E2E_P04ProtectStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P04ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P04 Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 4.	
Available via	E2E.h	

⌋

8.2.3.3 E2E_P04CheckStateType

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

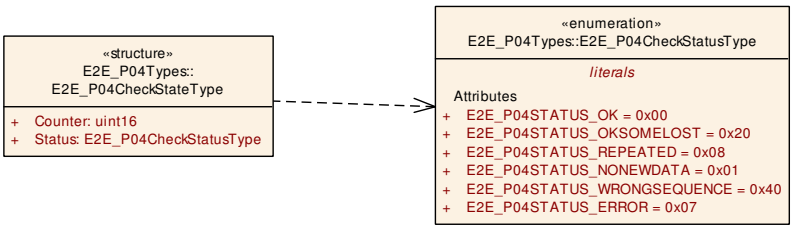


Figure 8.5: E2E Profile 4 check state

[SWS_E2E_00336] Definition of datatype E2E_P04CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P04CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P04CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint16
	Comment	Counter of the data in previous cycle.

▽

△

Description	State of the reception on one single Data protected with E2E Profile 4.
Available via	E2E.h

⌋

8.2.3.4 E2E_P04CheckStatusType

[SWS_E2E_00337] Definition of datatype E2E_P04CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P04CheckStatusType		
Kind	Enumeration		
Range	E2E_P04STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P04STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P04 STATUS_REPEATED.
	E2E_P04STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P04STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P04STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P04STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 4.		
Available via	E2E.h		

⌋

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

8.2.3.5 E2E_P04HeaderInformationType

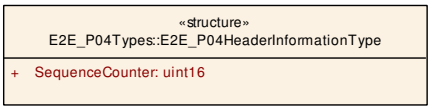


Figure 8.6: E2E Profile 04 header information type

[SWS_E2E_91103] Definition of datatype E2E_P04HeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P04HeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint16
	Comment	sequence counter of profile P04
Description	Data that can be retrieved from E2E header on receiver side in case E2E Profile 04 is used.	
Available via	E2E.h	

⌋

8.2.4 E2E Profile 4m types

8.2.4.1 E2E_P04mConfigType

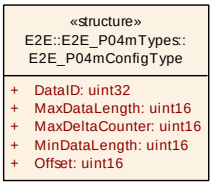


Figure 8.7: E2E Profile 4m configuration

[SWS_E2E_91021] Definition of datatype E2E_P04mConfigType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P04mConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint16

▽

△

	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(12*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint16
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00852]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint16
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00852]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint16
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 4m. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.4.2 E2E_P04mProtectStateType

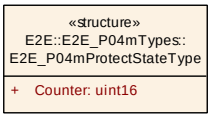


Figure 8.8: E2E Profile 4m Protect state type

[SWS_E2E_91020] Definition of datatype E2E_P04mProtectStateType

Upstream requirements: RS_E2E_08539

⌈

Name	E2E_P04mProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P04m Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 4m.	

▽

△

Available via	E2E.h

⌋

8.2.4.3 E2E_P04mCheckStateType

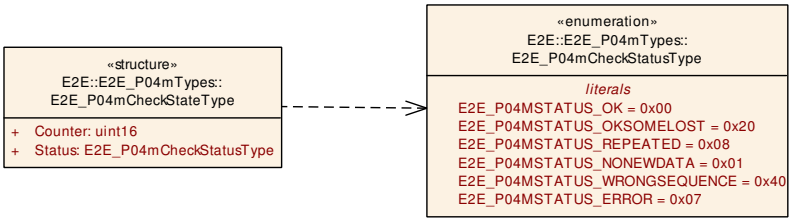


Figure 8.9: E2E Profile 4m check state

[SWS_E2E_91019] Definition of datatype E2E_P04mCheckStateType

Upstream requirements: RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P04mCheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P04mCheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint16
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 4m.	
Available via	E2E.h	

⌋

8.2.4.4 E2E_P04mCheckStatusType

[SWS_E2E_91022] Definition of datatype E2E_P04mCheckStatusType

Upstream requirements: RS_E2E_08534

⌈

Name	E2E_P04mCheckStatusType		
Kind	Enumeration		
Range	E2E_P04MSTATUS_OK	0x00	–

▽

△

	E2E_P04MSTATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P04 STATUS_REPEATED.
	E2E_P04MSTATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P04MSTATUS_REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P04MSTATUS_OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P04MSTATUS_WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 4m.		
Available via	E2E.h		

⌋

Note that the status E2E_P04MSTATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.4.5 E2E_P04mHeaderInformationType

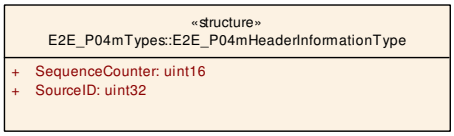


Figure 8.10: E2E Profile 04m header information type

[SWS_E2E_91109] Definition of datatype E2E_P04mHeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P04mHeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint16
	Comment	sequence counter of profile P04m
	SourceID	
	Type	uint32

▽

△

	Comment	sourceID
Description	Profile P04m specific data type containing the relevant profile specific header elements	
Available via	E2E.h	

⌋

8.2.5 E2E Profile 5 types

8.2.5.1 E2E_P05ConfigType

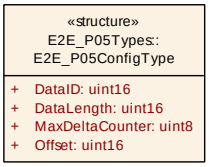


Figure 8.11: E2E Profile 5 configuration

[SWS_E2E_00437] Definition of datatype E2E_P05ConfigType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

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

▽

△

Available via	E2E.h

⌋

8.2.5.2 E2E_P05ProtectStateType

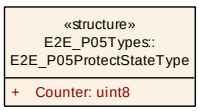


Figure 8.12: E2E Profile 5 Protect state type

[SWS_E2E_00438] Definition of datatype E2E_P05ProtectStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P05ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P05 Protect() is called, it increments the counter up to 0xFF.
Description	State of the sender for a Data protected with E2E Profile 5.	
Available via	E2E.h	

⌋

8.2.5.3 E2E_P05CheckStateType

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

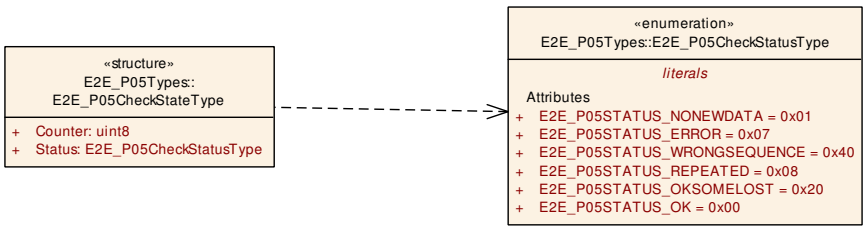


Figure 8.13: E2E Profile 5 Check state type

[SWS_E2E_00439] Definition of datatype E2E_P05CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P05CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P05CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint8
	Comment	Counter of the data in previous cycle.
Description	Description: State of the reception on one single Data protected with E2E Profile 5.	
Available via	E2E.h	

⌋

8.2.5.4 E2E_P05CheckStatusType

[SWS_E2E_00440] Definition of datatype E2E_P05CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P05CheckStatusType		
Kind	Enumeration		
Range	E2E_P05STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P05STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P05 STATUS_REPEATED.
	E2E_P05STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length).
	E2E_P05STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P05STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P05STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 5.		

▽

△

Available via	E2E.h

⌋

8.2.5.5 E2E_P05HeaderInformationType

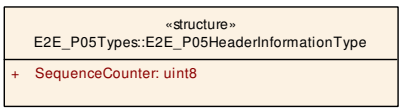


Figure 8.14: E2E Profile 05 header information type

[SWS_E2E_91104] Definition of datatype E2E_P05HeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P05HeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint8
	Comment	sequence counter of profile P05
Description	Data that can be retrieved from E2E header on receiver side in case E2E Profile 05 is used.	
Available via	E2E.h	

⌋

8.2.6 E2E Profile 6 types

8.2.6.1 E2E_P06ConfigType

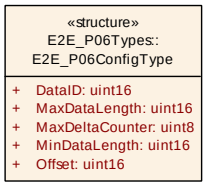


Figure 8.15: E2E Profile 6 configuration

[SWS_E2E_00441] Definition of datatype E2E_P06ConfigType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P06ConfigType	
Kind	Structure	
Elements	Offset	
	Type	uint16
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(5*8). Example: If Offset equals 8, then the high byte of the E2E Crc (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint16
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00657]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint16
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00657]). The value shall be a multiple of 8.
	DataID	
	Type	uint16
	Comment	A system-unique identifier of the Data
	MaxDeltaCounter	
	Type	uint8
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 6. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.6.2 E2E_P06ProtectStateType

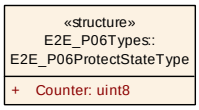


Figure 8.16: E2E Profile 6 Protect state type

[SWS_E2E_00443] Definition of datatype E2E_P06ProtectStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P06ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P06 Protect() is called, it increments the counter up to 0xFF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 6.	
Available via	E2E.h	

⌋

8.2.6.3 E2E_P06CheckStateType

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

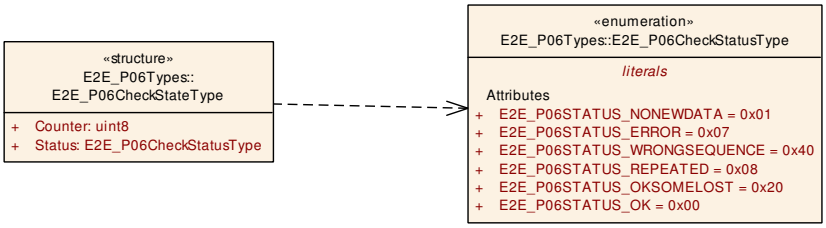


Figure 8.17: E2E Profile 6 Check state type

[SWS_E2E_00444] Definition of datatype E2E_P06CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

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

⌋

8.2.6.4 E2E_P06CheckStatusType

[SWS_E2E_00445] Definition of datatype E2E_P06CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P06CheckStatusType		
Kind	Enumeration		
Range	E2E_P06STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P06STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P06 STATUS_REPEATED.
	E2E_P06STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length).
	E2E_P06STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P06STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P06STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 6.		
Available via	E2E.h		

⌋

8.2.6.5 E2E_P06HeaderInformationType

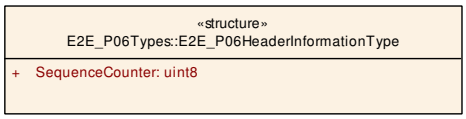


Figure 8.18: E2E Profile 06 header information type

[SWS_E2E_91105] Definition of datatype E2E_P06HeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P06HeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint8
	Comment	sequence counter of profile P06
Description	Data that can be retrieved from E2E header on receiver side in case E2E Profile 06 is used.	
Available via	E2E.h	

⌋

8.2.7 E2E Profile 7 types

8.2.7.1 E2E_P07ConfigType

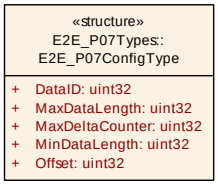


Figure 8.19: E2E Profile 7 configuration

[SWS_E2E_00544] Definition of datatype E2E_P07ConfigType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P07ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(20*8). Example: If Offset equals 8, then the first byte of the E2E Length (32 bit) is written to byte 1, the next byte is written to byte 2 and so on.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00660]). The value shall be a multiple of 8.

▽

△

	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00660]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint32
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 7. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.7.2 E2E_P07ProtectStateType

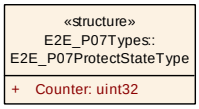


Figure 8.20: E2E Profile 7 Protect state type

[SWS_E2E_00545] Definition of datatype E2E_P07ProtectStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P07ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P07 Protect() is called, it increments the counter up to 0xFF'FF'FF'FF.
Description	State of the sender for a Data protected with E2E Profile 7.	
Available via	E2E.h	

⌋

8.2.7.3 E2E_P07CheckStateType

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

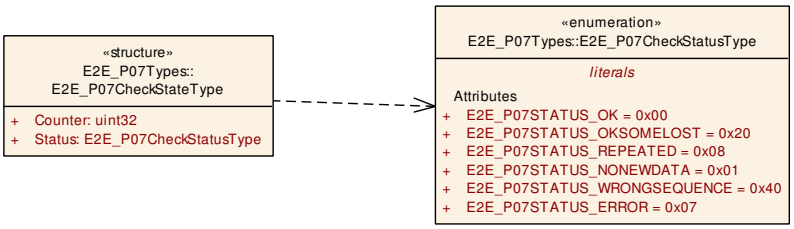


Figure 8.21: E2E Profile 7 Check state type

[SWS_E2E_00542] Definition of datatype E2E_P07CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

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

⌋

8.2.7.4 E2E_P07CheckStatusType

[SWS_E2E_00591] Definition of datatype E2E_P07CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P07CheckStatusType		
Kind	Enumeration		
Range	E2E_P07STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P07STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P07 STATUS_REPEATED.
	E2E_P07STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).

▽

△

	E2E_P07STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P07STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P07STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 7.		
Available via	E2E.h		

⌋

8.2.7.5 E2E_P07HeaderInformationType

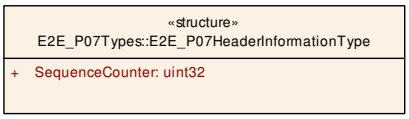


Figure 8.22: E2E Profile 07 header information type

[SWS_E2E_91106] Definition of datatype E2E_P07HeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P07HeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint32
	Comment	sequence counter of profile P07
Description	Data that can be retrieved from E2E header on receiver side in case E2E Profile 07 is used.	
Available via	E2E.h	

⌋

8.2.8 E2E Profile 7m types

8.2.8.1 E2E_P07mConfigType

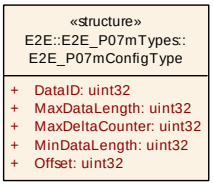


Figure 8.23: E2E Profile 7m configuration

[SWS_E2E_91010] Definition of datatype E2E_P07mConfigType

Upstream requirements: RS_E2E_08539

⌈

Name	E2E_P07mConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(24*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00851]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00851]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint32
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 7m. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.8.2 E2E_P07mProtectStateType

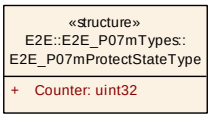


Figure 8.24: E2E Profile 7m Protect state type

[SWS_E2E_91011] Definition of datatype E2E_P07mProtectStateType

Upstream requirements: RS_E2E_08539

⌈

Name	E2E_P07mProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P07m Protect() is called, it increments the counter up to 0xFF'FF'FF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 7m.	
Available via	E2E.h	

⌋

8.2.8.3 E2E_P07mCheckStateType

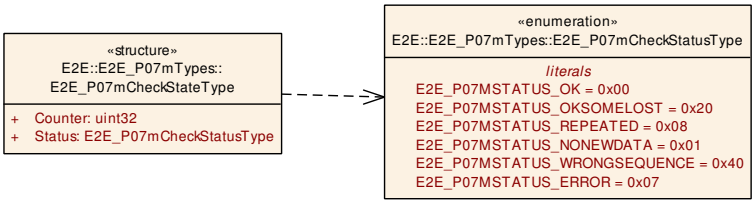


Figure 8.25: E2E Profile 7m Check state type

[SWS_E2E_91008] Definition of datatype E2E_P07mCheckStateType

Upstream requirements: RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P07mCheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P07mCheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint32
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 7m.	
Available via	E2E.h	

⌋

8.2.8.4 E2E_P07mCheckStatusType

[SWS_E2E_91009] Definition of datatype E2E_P07mCheckStatusType

Upstream requirements: RS_E2E_08534

⌈

Name	E2E_P07mCheckStatusType		
Kind	Enumeration		
Range	E2E_P07MSTATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P07MSTATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P07 STATUS_REPEATED.
	E2E_P07MSTATUS_ ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P07MSTATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P07MSTATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P07MSTATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 7m.		
Available via	E2E.h		

⌋

8.2.8.5 E2E_P07mHeaderInformationType

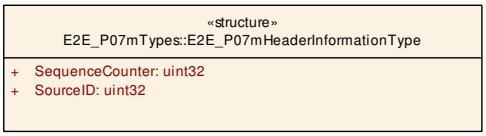


Figure 8.26: E2E Profile 07m header information type

[SWS_E2E_91110] Definition of datatype E2E_P07mHeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P07mHeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint32
	Comment	sequence counter of profile P07m
	SourceID	
	Type	uint32
	Comment	SourceID
Description	Profile P07m specific data type containing the relevant profile specific header elements	
Available via	E2E.h	

⌋

8.2.9 E2E Profile 8 types

8.2.9.1 E2E_P08ConfigType

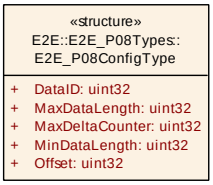


Figure 8.27: E2E Profile 08 configuration

[SWS_E2E_91033] Definition of datatype E2E_P08ConfigType

Upstream requirements: RS_E2E_08527

⌈

Name	E2E_P08ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(16*8). Example: If Offset equals 8, then the high byte of the E2E Length (32 bit) is written to byte 1, the next byte is written to byte 2 and so on.
	MinDataLength	
	Type	uint32

▽

△

	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00706]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00706]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint32
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 08. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.9.2 E2E_P08ProtectStateType

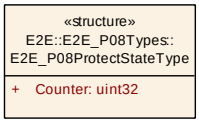


Figure 8.28: E2E Profile 08 Protect state type

[SWS_E2E_91042] Definition of datatype E2E_P08ProtectStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P08ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_ P08 Protect() is called, it increments the counter up to 0xFF'FF'FF'FF.
Description	State of the sender for a Data protected with E2E Profile 08.	
Available via	E2E.h	

⌋

8.2.9.3 E2E_P08CheckStateType

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

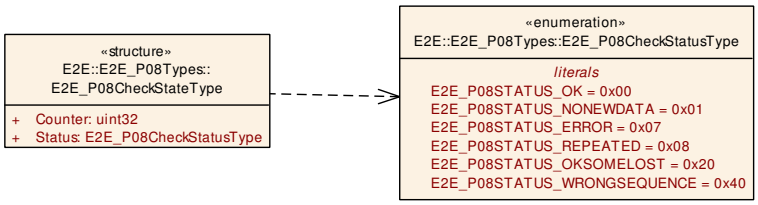


Figure 8.29: E2E Profile 08 Check state type

[SWS_E2E_91034] Definition of datatype E2E_P08CheckStateType

Upstream requirements: RS_E2E_08527

⌈

Name	E2E_P08CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P08CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint32
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 08	
Available via	E2E.h	

⌋

8.2.9.4 E2E_P08CheckStatusType

[SWS_E2E_91035] Definition of datatype E2E_P08CheckStatusType

Upstream requirements: RS_E2E_08527, RS_E2E_08534

⌈

Name	E2E_P08CheckStatusType		
Kind	Enumeration		
Range	E2E_P08STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P08STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_ P08 STATUS_REPEATED.
	E2E_P08STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).

▽

△

	E2E_P08STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P08STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P08STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 08.		
Available via	E2E.h		

⌋

Note that the status E2E_P08STATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.9.5 E2E_P08HeaderInformationType

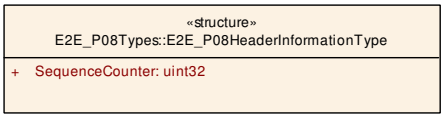


Figure 8.30: E2E Profile 08 header information type

[SWS_E2E_91107] Definition of datatype E2E_P08HeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P08HeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint32
	Comment	sequence counter of profile P08
Description	Data that can be retrieved from E2E header on receiver side in case E2E Profile 08 is used.	
Available via	E2E.h	

⌋

8.2.10 E2E Profile 8m types

8.2.10.1 E2E_P08mConfigType

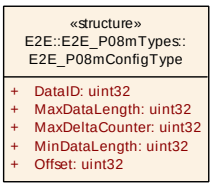


Figure 8.31: E2E Profile 08m configuration

[SWS_E2E_91073] Definition of datatype E2E_P08mConfigType

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Name	E2E_P08mConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_01154]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_01154]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint32
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(16*8). Example: If Offset equals 8, then the high byte of the E2E Length (32 bit) is written to byte 1, the next byte is written to byte 2 and so on.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 08m. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.10.2 E2E_P08mProtectStateType

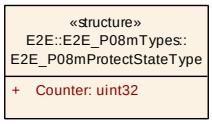


Figure 8.32: E2E Profile 08m Protect state type

[SWS_E2E_91074] Definition of datatype E2E_P08mProtectStateType

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Name	E2E_P08mProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint32
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P08mProtect() is called, it increments the counter up to 0xFF'FF'FF'FF.
Description	State of the sender for a Data protected with E2E Profile 08m.	
Available via	E2E.h	

⌋

8.2.10.3 E2E_P08mCheckStateType

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

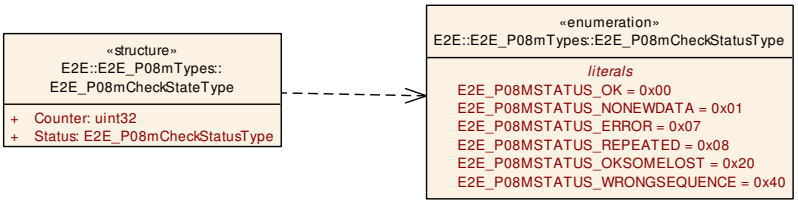


Figure 8.33: E2E Profile 08m Check state type

[SWS_E2E_91075] Definition of datatype E2E_P08mCheckStateType

Upstream requirements: RS_E2E_08527, RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P08mCheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P08mCheckStatusType

▽

△

	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint32
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 08m.	
Available via	E2E.h	

⌋

8.2.10.4 E2E_P08mCheckStatusType

[SWS_E2E_91076] Definition of datatype E2E_P08mCheckStatusType

Upstream requirements: RS_E2E_08527, RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P08mCheckStatusType		
Kind	Enumeration		
Range	E2E_P08MSTATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P08MSTATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P08 MSTATUS_REPEATED.
	E2E_P08MSTATUS_ ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P08MSTATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P08MSTATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P08MSTATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 08m.		
Available via	E2E.h		

⌋

Note that the status E2E_P08MSTATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.10.5 E2E_P08mHeaderInformationType

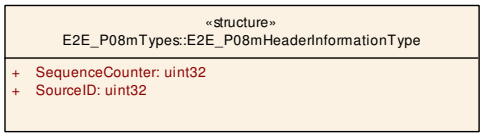


Figure 8.34: E2E Profile 08m header information type

[SWS_E2E_91111] Definition of datatype E2E_P08mHeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P08mHeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint32
	Comment	sequence counter of profile P08m
	SourceID	
	Type	uint32
	Comment	SourceID
Description	Profile P08m specific data type containing the relevant profile specific header elements	
Available via	E2E.h	

⌋

8.2.11 E2E Profile 11 types

8.2.11.1 E2E_P11ConfigType

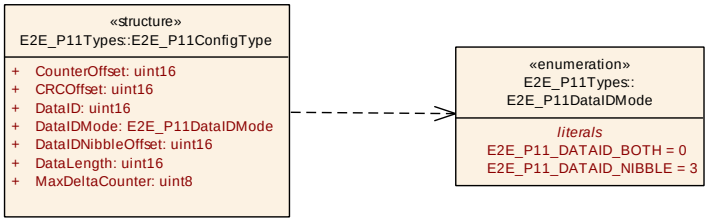


Figure 8.35: E2E Profile 11 configuration

[SWS_E2E_00565] Definition of datatype E2E_P11ConfigType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P11ConfigType
Kind	Structure
Elements	DataLength

▽

△

	Type	uint16
	Comment	Length of data, in bits. The value shall be a multiple of 8.
	DataID	
	Type	uint16
	Comment	A unique identifier, for protection against masquerading. There are some constraints on the selection of ID values, described in section "Configuration constraints on Data IDs".
	MaxDeltaCounter	
	Type	uint8
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
	DataIDMode	
	Type	E2E_P11DataIDMode
	Comment	–
	CRCOffset	
	Type	uint16
	Comment	Bit offset of CRC (i.e. since *Data) in MSB first order. In variants 1A and 1B, CRCOffset is 0. The offset shall be a multiple of 8.
	CounterOffset	
	Type	uint16
	Comment	Bit offset of Counter in MSB first order. In variants 1A and 1B, Counter Offset is 8. The offset shall be a multiple of 4.
	DataIDNibbleOffset	
	Type	uint16
	Comment	Bit offset of the low nibble of the high byte of Data ID. This parameter is used by E2E Library only if DataIDMode = E2E_P11_DATAID_NIBBLE (otherwise it is ignored by E2E Library). For DataIDMode different than E2E_P11_DATAID_NIBBLE, Data IDNibbleOffset shall be initialized to 0 (even if it is ignored by E2E Library).
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 11. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.11.2 E2E_P11DataIDMode

[SWS_E2E_00566] Definition of datatype E2E_P11DataIDMode

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P11DataIDMode		
Kind	Enumeration		
Range	E2E_P11_DATAID_BOTH	0	Two bytes are included in the CRC (double ID configuration) This is used in E2E variant 1A.
	E2E_P11_DATAID_NIBBLE	3	The low byte is included in the implicit CRC calculation, the low nibble of the high byte is transmitted along with the data (i.e. it is explicitly included), the high nibble of the high byte is not used. This is applicable for the IDs up to 12 bits. This is used in E2E variant 1C.
Description	The Data ID is two bytes long in E2E Profile 1. There are four inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC.		
Available via	E2E.h		

⌋

8.2.11.3 E2E_P11ProtectStateType

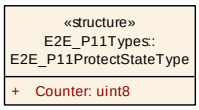


Figure 8.36: E2E Profile 11 Protect state type

[SWS_E2E_00567] Definition of datatype E2E_P11ProtectStateType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P11ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P11 Protect() is called, it increments the counter up to 0x0E.
Description	State of the sender for a Data protected with E2E Profile 11.	
Available via	E2E.h	

⌋

8.2.11.4 E2E_P11CheckStateType

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

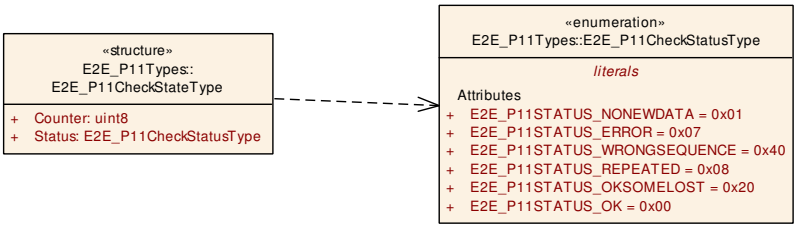


Figure 8.37: E2E Profile 11 Check state type

[SWS_E2E_00563] Definition of datatype E2E_P11CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P11CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P11CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint8
	Comment	Counter of the data in previous cycle.
Description	Description: State of the reception on one single Data protected with E2E Profile 11.	
Available via	E2E.h	

⌋

8.2.11.5 E2E_P11CheckStatusType

[SWS_E2E_00564] Definition of datatype E2E_P11CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P11CheckStatusType		
Kind	Enumeration		
Range	E2E_P11STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).

▽

△

	E2E_P11STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P11 STATUS_REPEATED.
	E2E_P11STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length).
	E2E_P11STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P11STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P11STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 11.		
Available via	E2E.h		

⌋

8.2.12 E2E Profile 22 types

8.2.12.1 E2E_P22ConfigType

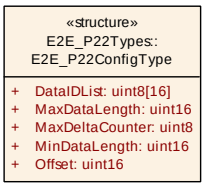


Figure 8.38: E2E Profile 22 configuration

[SWS_E2E_00571] Definition of datatype E2E_P22ConfigType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P22ConfigType	
Kind	Structure	
Elements	MinDataLength	
	Type	uint16
	Comment	Length of Data, in bits. The value shall be a multiple of 8 and MinData Length >= Offset +(2*8). The value shall be the same as MaxData Length.
	MaxDataLength	

▽

△

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

⌋

8.2.12.2 E2E_P22ProtectStateType

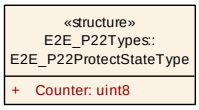


Figure 8.39: E2E Profile 22 Protect state type

[SWS_E2E_00570] Definition of datatype E2E_P22ProtectStateType

Upstream requirements: RS_E2E_08528

⌈

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

▽

△

Available via	E2E.h

⌋

8.2.12.3 E2E_P22CheckStateType

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

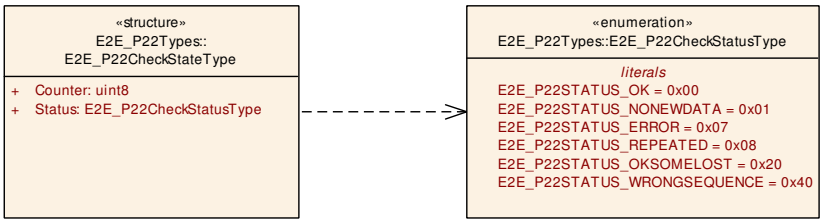


Figure 8.40: E2E Profile 22 Check state type

[SWS_E2E_00568] Definition of datatype E2E_P22CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

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

⌋

8.2.12.4 E2E_P22CheckStatusType

[SWS_E2E_00569] Definition of datatype E2E_P22CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P22CheckStatusType		
Kind	Enumeration		
Range	E2E_P22STATUS_OK	0x00	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
	E2E_P22STATUS_NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
	E2E_P22STATUS_ERROR	0x07	Error: The data has been received according to communication medium, but the CRC is incorrect.
	E2E_P22STATUS_REPEATED	0x08	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
	E2E_P22STATUS_OKSOMELOST	0x20	OK: The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter =Max DeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
	E2E_P22STATUS_WRONGSEQUENCE	0x40	Error: The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.
Description	Result of the verification of the Data in E2E Profile 22, determined by the Check function.		
Available via	E2E.h		

⌋

8.2.13 E2E Profile 44 types

8.2.13.1 E2E_P44ConfigType

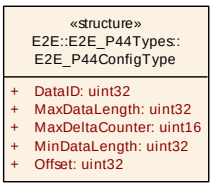


Figure 8.41: E2E Profile 44 configuration

[SWS_E2E_91023] Definition of datatype E2E_P44ConfigType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P44ConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(12*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_00735]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_00735]). The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint16
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 44. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.13.2 E2E_P44ProtectStateType

[SWS_E2E_91024] Definition of datatype E2E_P44ProtectStateType

Upstream requirements: RS_E2E_08528

⌈

Name	E2E_P44ProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P44 Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 44.	
Available via	E2E.h	

⌋

8.2.13.3 E2E_P44CheckStateType

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

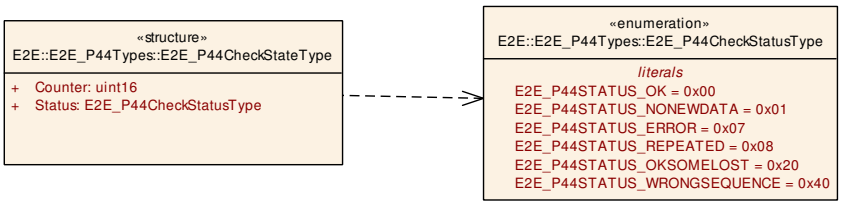


Figure 8.42: E2E Profile 44 Check state type

[SWS_E2E_91025] Definition of datatype E2E_P44CheckStateType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P44CheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P44CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint16
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 44.	

▽

△

Available via	E2E.h

⌋

8.2.13.4 E2E_P44CheckStatusType

[SWS_E2E_91026] Definition of datatype E2E_P44CheckStatusType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P44CheckStatusType		
Kind	Enumeration		
Range	E2E_P44STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P44STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_ P44 STATUS_REPEATED.
	E2E_P44STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P44STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P44STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P44STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 44.		
Available via	E2E.h		

⌋

Note that the status E2E_P44STATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.13.5 E2E_P44HeaderInformationType

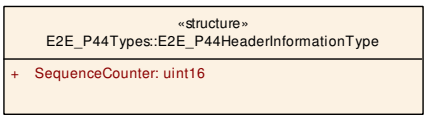


Figure 8.43: E2E Profile 44 header information type

[SWS_E2E_91108] Definition of datatype E2E_P44HeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P44HeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint16
	Comment	sequence counter of profile P44
Description	Data that can be retrieved from E2E header on receiver side in case E2E Profile 44 is used.	
Available via	E2E.h	

⌋

8.2.14 E2E Profile 44m types

8.2.14.1 E2E_P44mConfigType

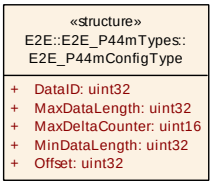


Figure 8.44: E2E Profile 44m configuration

[SWS_E2E_91077] Definition of datatype E2E_P44mConfigType

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Name	E2E_P44mConfigType	
Kind	Structure	
Elements	DataID	
	Type	uint32
	Comment	A system-unique identifier of the Data.
	MinDataLength	
	Type	uint32
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength (see [PRS_E2E_01202]). The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint32
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength (see [PRS_E2E_01202]). The value shall be a multiple of 8.
	MaxDeltaCounter	

▽

△

	Type	uint16
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
	Offset	
	Type	uint32
	Comment	Bit offset of the first bit of the E2E header from the beginning of the Data (bit numbering: bit 0 is the least important). The offset shall be a multiple of 8 and 0 <= Offset <= MaxDataLength-(12*8). Example: If Offset equals 8, then the high byte of the E2E Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 44m. For each transmitted Data, there is an instance of this typedef.	
Available via	E2E.h	

⌋

8.2.14.2 E2E_ P44mProtectStateType

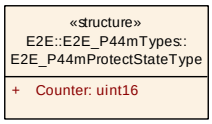


Figure 8.45: E2E Profile 44m Protect state type

[SWS_E2E_91078] Definition of datatype E2E_P44mProtectStateType

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Name	E2E_P44mProtectStateType	
Kind	Structure	
Elements	Counter	
	Type	uint16
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_P44m Protect() is called, it increments the counter up to 0xFF'FF. After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 44m.	
Available via	E2E.h	

⌋

8.2.14.3 E2E_P44mCheckStateType

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

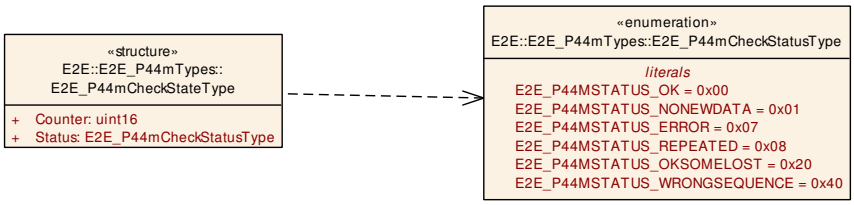


Figure 8.46: E2E Profile 44m Check state type

[SWS_E2E_91079] Definition of datatype E2E_P44mCheckStateType

Upstream requirements: RS_E2E_08527, RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P44mCheckStateType	
Kind	Structure	
Elements	Status	
	Type	E2E_P44mCheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	
	Type	uint16
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 44m.	
Available via	E2E.h	

⌋

8.2.14.4 E2E_P44mCheckStatusType

[SWS_E2E_91080] Definition of datatype E2E_P44mCheckStatusType

Upstream requirements: RS_E2E_08527, RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P44mCheckStatusType		
Kind	Enumeration		
Range	E2E_P44MSTATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P44MSTATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is not available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_P44 MSTATUS_REPEATED.
	E2E_P44MSTATUS_ ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).

▽

△

	E2E_P44MSTATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P44MSTATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P44MSTATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 44m.		
Available via	E2E.h		

⌋

Note that the status E2E_P44MSTATUS_ERROR is new (with respect to E2E Profiles 1 and 2).

8.2.14.5 E2E_P44mHeaderInformationType

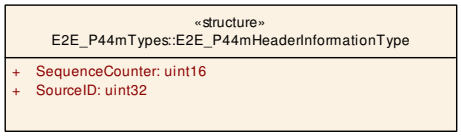


Figure 8.47: E2E Profile 44m header information type

[SWS_E2E_91112] Definition of datatype E2E_P44mHeaderInformationType

Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P44mHeaderInformationType	
Kind	Structure	
Elements	SequenceCounter	
	Type	uint16
	Comment	sequence counter of profile P44m
	SourceID	
	Type	uint32
	Comment	SourceID
Description	Profile P44m specific data type containing the relevant profile specific header elements	
Available via	E2E.h	

⌋

8.2.15 E2E Profile 76 Types

8.2.15.1 E2E_P76ConfigType

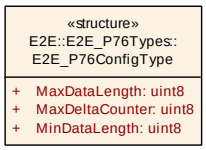


Figure 8.48: E2E Profile 76 configuration

[SWS_E2E_00616] Definition of datatype E2E_P76ConfigType

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P76ConfigType (draft)	
Kind	Structure	
Elements	MinDataLength	
	Type	uint8
	Comment	Minimal length of Data, in bits. E2E checks that DataLength is >= Min DataLength. The value shall be a multiple of 8.
	MaxDataLength	
	Type	uint8
	Comment	Maximal length of Data, in bits. E2E checks that DataLength is <= Max DataLength. The value shall be a multiple of 8.
	MaxDeltaCounter	
	Type	uint8
	Comment	Maximum allowed gap between two counter values of two consecutively received valid Data. For example, if the receiver gets Data with counter 1 and MaxDeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
Description	Configuration of transmitted Data (Data Element or I-PDU), for E2E Profile 76. For each transmitted Data, there is an instance of this typedef. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.2.15.2 E2E_P76ProtectStateType

[SWS_E2E_00619] Definition of datatype E2E_P76ProtectStateType

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈

Name	E2E_P76ProtectStateType (draft)	
Kind	Structure	
Elements	Counter	
	Type	uint8
	Comment	Counter to be used for protecting the next Data. The initial value is 0, which means that in the first cycle, Counter is 0. Each time E2E_ P76 Protect() is called, it increments the counter up to 0x1F (31). After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.
Description	State of the sender for a Data protected with E2E Profile 76. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.2.15.3 E2E_P76CheckStateType

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

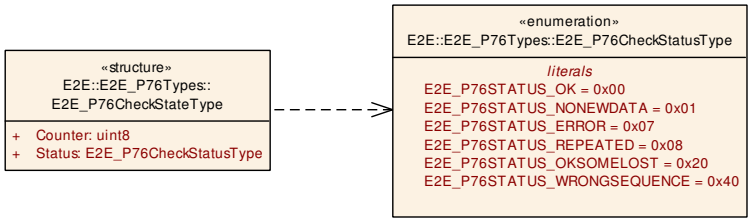


Figure 8.49: E2E Profile 76 check state

[SWS_E2E_00617] Definition of datatype E2E_P76CheckStateType

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08534

⌈

Name	E2E_P76CheckStateType (draft)	
Kind	Structure	
Elements	Status	
	Type	E2E_P76CheckStatusType
	Comment	Result of the verification of the Data in this cycle, determined by the Check function.
	Counter	

▽

△

	Type	uint8
	Comment	Counter of the data in previous cycle.
Description	State of the reception on one single Data protected with E2E Profile 76. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.2.15.4 E2E_P76CheckStatusType

[SWS_E2E_00618] Definition of datatype E2E_P76CheckStatusType

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08534

⌈

Name	E2E_P76CheckStatusType (draft)		
Kind	Enumeration		
Range	E2E_P76STATUS_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
	E2E_P76STATUS_ NONEWDATA	0x01	Error: the Check function has been invoked but no new Data is available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed. This may be considered similar to E2E_ P76 STATUS_REPEATED.
	E2E_P76STATUS_ERROR	0x07	Error: error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
	E2E_P76STATUS_ REPEATED	0x08	Error: the checks of the Data in this cycle were successful, with the exception of the repetition.
	E2E_P76STATUS_ OKSOMELOST	0x20	OK: the checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
	E2E_P76STATUS_ WRONGSEQUENCE	0x40	Error: the checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
Description	Status of the reception on one single Data in one cycle, protected with E2E Profile 76. **Tags:** atp.Status=draft		
Available via	E2E.h		

⌋

8.2.16 E2E state machine types

8.2.16.1 E2E_PCheckStatusType

[SWS_E2E_00347] Definition of datatype E2E_PCheckStatusType

Upstream requirements: RS_E2E_08548

⌈

Name	E2E_PCheckStatusType		
Kind	Enumeration		
Range	E2E_P_OK	0x00	OK: the checks of the Data in this cycle were successful (including counter check).
	E2E_P_REPEATED	0x01	Data has a repeated counter.
	E2E_P_ WRONGSEQUENCE	0x02	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
	E2E_P_ERROR	0x03	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong Data ID) or the return of the check function was not OK.
	E2E_P_NOTAVAILABLE	0x04	No value has been received yet (e.g. during initialization). This is used as the initialization value for the buffer, it is not returned by any E2E profile.
	E2E_P_NONEWDATA	0x05	No new data is available.
	reserved	0x07, 0x0F	reserved for runtime errors (shall not be used for any status in future).
Description	Profile-independent status of the reception on one single Data in one cycle.		
Available via	E2E.h		

⌋

8.2.16.2 E2E_SMConfigType

[SWS_E2E_00342] Definition of datatype E2E_SMConfigType

Upstream requirements: RS_E2E_08548

⌈

Name	E2E_SMConfigType	
Kind	Structure	
Elements	WindowSizeValid	
	Type	uint8
	Comment	Size of the monitoring window for the state machine during state VALID.
	MinOkStateInit	
	Type	uint8
	Comment	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_ SM_INIT) required to change to state E2E_SM_VALID.

▽

△

	MaxErrorStateInit	
	Type	uint8
	Comment	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INIT).
	MinOkStateValid	
	Type	uint8
	Comment	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_VALID) required to keep in state E2E_SM_VALID.
	MaxErrorStateValid	
	Type	uint8
	Comment	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_VALID).
	MinOkStateInvalid	
	Type	uint8
	Comment	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined within the last WindowSize checks (for the state E2E_SM_INVALID) required to change to state E2E_SM_VALID.
	MaxErrorStateInvalid	
	Type	uint8
	Comment	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INVALID).
	WindowSizeInit	
	Type	uint8
	Comment	Size of the monitoring windows for the state machine during state INIT.
	WindowSizeInvalid	
	Type	uint8
	Comment	Size of the monitoring window for the state machine during state INVALID.
	ClearToInvalid	
	Type	boolean
	Comment	Clear monitoring window data on transition to state INVALID.
	transitToInvalidExtended	
	Type	boolean
	Comment	Restrict/allow tranistion from states INIT/NODATA to INVALID state.
Description	Configuration of a communication channel for exchanging Data.	
Available via	E2E.h	

⌋

8.2.16.3 E2E_SMCheckStateType

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

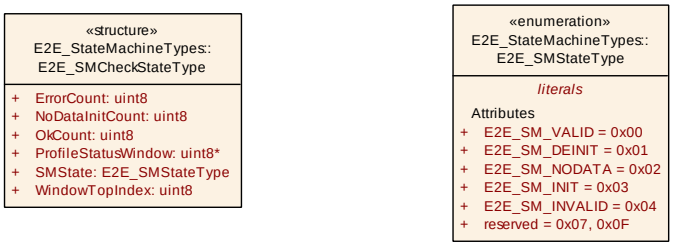


Figure 8.50: E2E SM check state

[SWS_E2E_00343] Definition of datatype E2E_SMCheckStateType

Upstream requirements: RS_E2E_08548

⌈

Name	E2E_SMCheckStateType	
Kind	Structure	
Elements	ProfileStatusWindow	
	Type	uint8*
	Comment	Pointer to an array, in which the ProfileStatus-es of the last E2 E-checks are stored. The array size shall be WindowSize
	WindowTopIndex	
	Type	uint8
	Comment	index in the array, at which the next ProfileStatus is to be written.
	OkCount	
	Type	uint8
	Comment	Count of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks.
	ErrorCount	
	Type	uint8
	Comment	Count of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks.
	SMState	
	Type	E2E_SMStateType
	Comment	The current state in the state machine. The value is not explicitly used in the pseudocode of the state machine, because it is expressed in UML as UML states.
	NoDataInitCount	
	Type	uint8
	Comment	Count of checks in the state E2E_SM_NODATA or E2E_SM_INIT without transitioning to the state E2E_SM_VALID. Length of this counter is less or equal than WindowSizeValid.
Description	State of the protection of a communication channel.	
Available via	E2E.h	

⌋

8.2.16.4 E2E_SMStateType

[SWS_E2E_00344] Definition of datatype E2E_SMStateType

Upstream requirements: RS_E2E_08548

⌈

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

⌋

8.3 Function definitions

This chapter contains all APIs provided by the module. These can be implemented as:

- Functions
- Inline functions
- Macros

8.3.1 E2E Profile 1 routines

8.3.1.1 E2E_P01Protect

[SWS_E2E_00166] Definition of API function E2E_P01Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P01Protect	
Syntax	`Std_ReturnType E2E_P01Protect ( const E2E_P01ConfigType* ConfigPtr, E2E_P01ProtectStateType* StatePtr, uint8* DataPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 1. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

⌋

8.3.1.2 E2E_P01ProtectInit

[SWS_E2E_00385] Definition of API function E2E_P01ProtectInit

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

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

▽

△

Description	Initializes the protection state.
Available via	E2E.h

⌋

[SWS_E2E_00386]

Upstream requirements: RS_E2E_08528

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

8.3.1.3 E2E_P01Forward

[SWS_E2E_00588] Definition of API function E2E_P01Forward

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P01Forward (draft)	
Syntax	`Std_ReturnType E2E_P01Forward ( const E2E_P01ConfigType* ConfigPtr, E2E_PCheckStatusType ForwardStatus, E2E_P01ProtectStateType* StatePtr, uint8* DataPtr )`	
Service ID [hex]	0x38	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 01. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.1.4 E2E_P01Check

[SWS_E2E_00158] Definition of API function E2E_P01Check

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P01Check	
Syntax	`Std_ReturnType E2E_P01Check ( const E2E_P01ConfigType* Config, E2E_P01CheckStateType* State, const uint8* Data )`	
Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Config	Pointer to static configuration.
	Data	Pointer to received data.
Parameters (inout)	State	Pointer to port/data communication state.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

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

⌋

8.3.1.5 E2E_P01CheckInit

[SWS_E2E_00390] Definition of API function E2E_P01CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

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

▽

△

Available via	E2E.h

⌋

[SWS_E2E_00389]

Upstream requirements: RS_E2E_08528

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

- LastValidCounter = 0
- MaxDeltaCounter = 0
- WaitForFirstData = TRUE
- NewDataAvailable = TRUE
- LostData = 0
- Status = E2E_P01STATUS_NONEWDATA
- NoNewOrRepeatedDataCounter = 0
- SyncCounter = 0.

⌋

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

8.3.1.6 E2E_P01MapStatusToSM

[SWS_E2E_00382] Definition of API function E2E_P01MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P01MapStatusToSM	
Syntax	`E2E_PCheckStatusType E2E_P01MapStatusToSM ( Std_ReturnType CheckReturn, E2E_P01CheckStatusType Status, boolean profileBehavior )`	
Service ID [hex]	0x1d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P01Check function
	Status	Status determined by E2E_P01Check function

▽

△

	profileBehavior	FALSE: check has the legacy behavior, before R4.2 TRUE: check behaves like new P4/P5/P6 profiles introduced in R4.2
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 1 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 1 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

This represents the R4.2 behavior:

[SWS_E2E_00383]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn == E2E_E_OK and ProfileBehavior == TRUE, then the function E2E_P01MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00602]⌋

[SWS_E2E_00602] Mapping of Profile 1

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈

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

⌋

This represents the pre-R4.2 behavior.

[SWS_E2E_00476]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn == E2E_E_OK and ProfileBehavior == FALSE, then the function E2E_P01MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00605]⌋

[SWS_E2E_00605]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈

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

⌋

[SWS_E2E_00384]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

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

8.3.2 E2E Profile 2 routines

8.3.2.1 E2E_P02Protect

[SWS_E2E_00160] Definition of API function E2E_P02Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P02Protect	

Syntax	`Std_ReturnType E2E_P02Protect (`
	`  const E2E_P02ConfigType* ConfigPtr,`
	`  E2E_P02ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to the data to be protected.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

▽

△

Description	Protects the array/buffer to be transmitted using the E2E profile 2. This includes checksum calculation, handling of sequence counter and Data ID.
Available via	E2E.h

⌋

8.3.2.2 E2E_P02ProtectInit

[SWS_E2E_00387] Definition of API function E2E_P02ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

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

⌋

[SWS_E2E_00388]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P02ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.2.3 E2E_P02Forward

[SWS_E2E_00583] Definition of API function E2E_P02Forward

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P02Forward (draft)	

Syntax	`Std_ReturnType E2E_P02Forward (`
	`  const E2E_P02ConfigType* ConfigPtr,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P02ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x32	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 02. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.2.4 E2E_P02Check

[SWS_E2E_00161] Definition of API function E2E_P02Check

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P02Check

Syntax	`Std_ReturnType E2E_P02Check (`
	`  const E2E_P02ConfigType* ConfigPtr,`
	`  E2E_P02CheckStateType* StatePtr,`
	`  const uint8* DataPtr`
	`)`

Service ID [hex]	0x04
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	–
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

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

⌋

8.3.2.5 E2E_P02CheckInit

[SWS_E2E_00391] Definition of API function E2E_P02CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P02CheckInit	

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

Service ID [hex]	0x1f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00392]

Upstream requirements: RS_E2E_08528

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

- LastValidCounter = 0
- MaxDeltaCounter = 0
- WaitForFirstData = TRUE
- NewDataAvailable = TRUE

- LostData = 0
- Status = E2E_P02STATUS_NONEWDATA
- NoNewOrRepeatedDataCounter = 0
- SyncCounter = 0.

⌋

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

8.3.2.6 E2E_P02MapStatusToSM

[SWS_E2E_00379] Definition of API function E2E_P02MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

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

⌋

This represents the R4.2 behavior:

[SWS_E2E_00380]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn == E2E_E_OKand ProfileBehavior == 1, then the function E2E_P02MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00601]⌋

[SWS_E2E_00601] Mapping of Profile 1

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈

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

⌋

This represents the pre-R4.2 behavior.

[SWS_E2E_00477]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn == E2E_E_OK and ProfileBehavior == 0, then the function E2E_P02MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00606]⌋

[SWS_E2E_00606] Mapping of profile 02

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈

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

⌋

[SWS_E2E_00381]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

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

8.3.3 E2E Profile 4 routines

8.3.3.1 E2E_P04Protect

[SWS_E2E_00338] Definition of API function E2E_P04Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04Protect	
Syntax	`Std_ReturnType E2E_P04Protect ( const E2E_P04ConfigType* ConfigPtr, E2E_P04ProtectStateType* StatePtr, uint8* DataPtr, uint16 Length )`	
Service ID [hex]	0x70	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 4. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

⌋

8.3.3.2 E2E_P04ProtectInit

[SWS_E2E_00373] Definition of API function E2E_P04ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04ProtectInit
Syntax	`Std_ReturnType E2E_P04ProtectInit ( E2E_P04ProtectStateType* StatePtr )`
Service ID [hex]	0x22
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_00377]

Upstream requirements: RS_E2E_08539

⌈In case State is NULL, E2E_P04ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.3.3 E2E_P04Forward

[SWS_E2E_00584] Definition of API function E2E_P04Forward

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P04Forward (draft)	

Syntax	`Std_ReturnType E2E_P04Forward (`
	`  const E2E_P04ConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P04ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x33	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 04. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.3.4 E2E_P04Check

[SWS_E2E_00339] Definition of API function E2E_P04Check

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

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

⌋

8.3.3.5 E2E_P04CheckInit

[SWS_E2E_00350] Definition of API function E2E_P04CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04CheckInit	
Syntax	`Std_ReturnType E2E_P04CheckInit ( E2E_P04CheckStateType* StatePtr )`	
Service ID [hex]	0x24	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.

▽

△

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00378]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

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

1. Counter to 0xFF'FF.
2. Status to E2E_P04STATUS_ERROR.

⌋

8.3.3.6 E2E_P04MapStatusToSM

[SWS_E2E_00349] Definition of API function E2E_P04MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04MapStatusToSM	

Syntax	`E2E_PCheckStatusType E2E_P04MapStatusToSM (`
	`  Std_ReturnType CheckReturn,`
	`  E2E_P04CheckStatusType Status`
	`)`	

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

⌋

[SWS_E2E_00351]

Upstream requirements: RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P04MapStatusToSMshall return the values depending on the value of Status: See [SWS_E2E_00600]⌋

[SWS_E2E_00600] Mapping of profile 04

Upstream requirements: RS_E2E_08548

⌈

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

⌋

[SWS_E2E_00352]

Upstream requirements: RS_E2E_08548

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

8.3.3.7 E2E_P04GetHeaderInfo

[SWS_E2E_91113] Definition of API function E2E_P04GetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P04GetHeaderInfo	
Syntax	`Std_ReturnType E2E_P04GetHeaderInfo ( const E2E_P04ConfigType* ConfigPtr, const uint8* DataPtr, uint16 Length, E2E_P04HeaderInformationType* HeaderInfoPtr )`	
Service ID [hex]	0x88	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 04. This includes the SequenceCounter.	
Available via	E2E.h	

⌋

8.3.4 E2E Profile 4m routines

8.3.4.1 E2E_P04mProtect

[SWS_E2E_91005] Definition of API function E2E_P04mProtect

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04mProtect	

Syntax	`Std_ReturnType E2E_P04mProtect (`
	`  const E2E_P04mConfigType* ConfigPtr,`
	`  E2E_P04mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID.	
Available via	E2E.h	

⌋

8.3.4.2 E2E_P04mProtectInit

[SWS_E2E_91006] Definition of API function E2E_P04mProtectInit

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04mProtectInit

Syntax	`Std_ReturnType E2E_P04mProtectInit (`
	`  E2E_P04mProtectStateType* StatePtr`
	`)`

▽

△

Service ID [hex]	0x47	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_00592]

Upstream requirements: RS_E2E_08539

⌈In case State is NULL, E2E_P04mProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.4.3 E2E_P04mForward

[SWS_E2E_91007] Definition of API function E2E_P04mForward

Status: DRAFT

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P04mForward (draft)	

Syntax	`Std_ReturnType E2E_P04mForward (`
	`  const E2E_P04mConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P04mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)

▽

△

Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.4.4 E2E_P04mSourceCheck

[SWS_E2E_91002] Definition of API function E2E_P04mSourceCheck

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04mSourceCheck	

Syntax	`Std_ReturnType E2E_P04mSourceCheck (`
	`  const E2E_P04mConfigType* ConfigPtr,`
	`  E2E_P04mCheckStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x43	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

▽

△

Description	Checks the Data received using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
Available via	E2E.h

⌋

8.3.4.5 E2E_P04mSinkCheck

[SWS_E2E_91003] Definition of API function E2E_P04mSinkCheck

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04mSinkCheck	

Syntax	`Std_ReturnType E2E_P04mSinkCheck (`
	`  const E2E_P04mConfigType* ConfigPtr,`
	`  E2E_P04mCheckStateType StatePtr,`
	`  uint32* SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x44	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	SourceID	A system-unique identifier of the Data Source.

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 4m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).	
Available via	E2E.h	

⌋

8.3.4.6 E2E_P04mCheckInit

[SWS_E2E_91001] Definition of API function E2E_P04mCheckInit

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04mCheckInit	
Syntax	`Std_ReturnType E2E_P04mCheckInit ( E2E_P04mCheckStateType* StatePtr )`	
Service ID [hex]	0x42	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00593]

Upstream requirements: RS_E2E_08539

⌈In case State is NULL, E2E_P04mCheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF.
2. Status to E2E_P04MSTATUS_ERROR.

⌋

8.3.4.7 E2E_P04mMapStatusToSM

[SWS_E2E_91004] Definition of API function E2E_P04mMapStatusToSM

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P04mMapStatusToSM
Syntax	`E2E_PCheckStatusType E2E_P04mMapStatusToSM ( Std_ReturnType CheckReturn, E2E_P04mCheckStatusType Status )`
Service ID [hex]	0x45
Sync/Async	Synchronous

▽

△

Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P04mSinkCheck/E2E_P04mSource Check function
	Status	Status determined by E2E_P04mSinkCheck/E2E_P04mSource Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 4m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 4m delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_00594]

Upstream requirements: RS_E2E_08539

⌈If CheckReturn = E2E_E_OK, then the function E2E_P04mMapStatusToSMshall return the values depending on the value of Status: See [SWS_E2E_00610]⌋

[SWS_E2E_00610] Mapping of profile 04m

Upstream requirements: RS_E2E_08539

⌈

Status	Return value
E2E_P04MSTATUS_OK or E2E_P04MSTATUS_ OKSOMELOST	E2E_P_OK
E2E_P04MSTATUS_ERROR	E2E_P_ERROR
E2E_P04MSTATUS_REPEATED	E2E_P_REPEATED
E2E_P04MSTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P04MSTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_00595]

Upstream requirements: RS_E2E_08539

⌈If CheckReturn != E2E_E_OK, then the function E2E_P04mMapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.4.8 E2E_P04mGetHeaderInfo

[SWS_E2E_91119] Definition of API function E2E_P04mGetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P04mGetHeaderInfo	

Syntax	`Std_ReturnType E2E_P04mGetHeaderInfo (`
	`  const E2E_P04mConfigType* ConfigPtr,`
	`  const uint8* DataPtr,`
	`  uint16 Length,`
	`  E2E_P04mHeaderInformationType* HeaderInfoPtr`
	`)`	

Service ID [hex]	0x8e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter and Source ID) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 04m. This includes SequenceCounter and SourceID.	
Available via	E2E.h	

⌋

8.3.5 E2E Profile 5 routines

8.3.5.1 E2E_P05Protect

[SWS_E2E_00446] Definition of API function E2E_P05Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P05Protect	

Syntax	`Std_ReturnType E2E_P05Protect (`
	`  const E2E_P05ConfigType* ConfigPtr,`
	`  E2E_P05ProtectStateType* StatePtr,`
	`  uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x26	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.

▽

△

	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 5. This includes checksum calculation, handling of counter.	
Available via	E2E.h	

⌋

8.3.5.2 E2E_P05ProtectInit

[SWS_E2E_00447] Definition of API function E2E_P05ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P05ProtectInit	

Syntax	`Std_ReturnType E2E_P05ProtectInit (`
	`  E2E_P05ProtectStateType* StatePtr`
	`)`	

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

⌋

[SWS_E2E_00448]

Upstream requirements: RS_E2E_08539

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

8.3.5.3 E2E_P05Forward

[SWS_E2E_00585] Definition of API function E2E_P05Forward

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P05Forward (draft)	

Syntax	`Std_ReturnType E2E_P05Forward (`
	`  const E2E_P05ConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P05ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x34	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 05. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.5.4 E2E_P05Check

[SWS_E2E_00449] Definition of API function E2E_P05Check

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P05Check

Syntax	`Std_ReturnType E2E_P05Check (`
	`  const E2E_P05ConfigType* ConfigPtr,`
	`  E2E_P05CheckStateType* StatePtr,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`

Service ID [hex]	0x28

▽

△

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 5. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

8.3.5.5 E2E_P05CheckInit

[SWS_E2E_00450] Definition of API function E2E_P05CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P05CheckInit	

Syntax	`Std_ReturnType E2E_P05CheckInit (`
	`  E2E_P05CheckStateType* StatePtr`
	`)`	

Service ID [hex]	0x29	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00451]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

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

1. Counter to 0xFF

2. Status to E2E_P05STATUS_ERROR.

⌋

8.3.5.6 E2E_P05MapStatusToSM

[SWS_E2E_00452] Definition of API function E2E_P05MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

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

⌋

[SWS_E2E_00453]

Upstream requirements: RS_E2E_08539, RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P05MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00603]⌋

[SWS_E2E_00603] Mapping of profile 05

Upstream requirements: RS_E2E_08539, RS_E2E_08548

⌈

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

⌋

[SWS_E2E_00454]

Upstream requirements: RS_E2E_08548

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

8.3.5.7 E2E_P05GetHeaderInfo

[SWS_E2E_91114] Definition of API function E2E_P05GetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P05GetHeaderInfo	
Syntax	`Std_ReturnType E2E_P05GetHeaderInfo ( const E2E_P05ConfigType* ConfigPtr, const uint8* DataPtr, uint8 Length, E2E_P05HeaderInformationType* HeaderInfoPtr )`	
Service ID [hex]	0x89	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 05. This includes the SequenceCounter.	
Available via	E2E.h	

⌋

8.3.6 E2E Profile 6 routines

8.3.6.1 E2E_P06Protect

[SWS_E2E_00393] Definition of API function E2E_P06Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

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

⌋

8.3.6.2 E2E_P06ProtectInit

[SWS_E2E_00455] Definition of API function E2E_P06ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P06ProtectInit
Syntax	`Std_ReturnType E2E_P06ProtectInit ( E2E_P06ProtectStateType* StatePtr )`
Service ID [hex]	0x2c
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_00456]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

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

8.3.6.3 E2E_P06Forward

[SWS_E2E_00586] Definition of API function E2E_P06Forward

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P06Forward (draft)	

Syntax	`Std_ReturnType E2E_P06Forward (`
	`  const E2E_P06ConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P06ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x35	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 06. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.6.4 E2E_P06Check

[SWS_E2E_00457] Definition of API function E2E_P06Check

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P06Check	
Syntax	`Std_ReturnType E2E_P06Check ( const E2E_P06ConfigType* ConfigPtr, E2E_P06CheckStateType* StatePtr, const uint8* DataPtr, uint16 Length )`	
Service ID [hex]	0x2d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 6. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

8.3.6.5 E2E_P06CheckInit

[SWS_E2E_00458] Definition of API function E2E_P06CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P06CheckInit	
Syntax	`Std_ReturnType E2E_P06CheckInit ( E2E_P06CheckStateType* StatePtr )`	
Service ID [hex]	0x2e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.

▽

△

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00459]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

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

1. Counter to 0xFF
2. Status to E2E_P06STATUS_ERROR.

⌋

8.3.6.6 E2E_P06MapStatusToSM

[SWS_E2E_00460] Definition of API function E2E_P06MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

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

⌋

[SWS_E2E_00461]

Upstream requirements: RS_E2E_08539, RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P06MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00604]⌋

[SWS_E2E_00604] Mapping of profile 06

Upstream requirements: RS_E2E_08539, RS_E2E_08548

⌈

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

⌋

[SWS_E2E_00462]

Upstream requirements: RS_E2E_08548

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

8.3.6.7 E2E_P06GetHeaderInfo

[SWS_E2E_91115] Definition of API function E2E_P06GetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P06GetHeaderInfo	

Syntax	`Std_ReturnType E2E_P06GetHeaderInfo (`
	`  const E2E_P06ConfigType* ConfigPtr,`
	`  const uint8* DataPtr,`
	`  uint8 Length,`
	`  E2E_P06HeaderInformationType* HeaderInfoPtr`
	`)`	

Service ID [hex]	0x8a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 06. This includes the SequenceCounter.	
Available via	E2E.h	

⌋

8.3.7 E2E Profile 7 routines

8.3.7.1 E2E_P07Protect

[SWS_E2E_00546] Definition of API function E2E_P07Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07Protect	
Syntax	`Std_ReturnType E2E_P07Protect ( const E2E_P07ConfigType* ConfigPtr, E2E_P07ProtectStateType* StatePtr, uint8* DataPtr, uint32 Length )`	
Service ID [hex]	0x21	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

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

⌋

8.3.7.2 E2E_P07ProtectInit

[SWS_E2E_00547] Definition of API function E2E_P07ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07ProtectInit
Syntax	`Std_ReturnType E2E_P07ProtectInit ( E2E_P07ProtectStateType* StatePtr )`
Service ID [hex]	0x77
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

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

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_00551]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P07ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.7.3 E2E_P07Forward

[SWS_E2E_00590] Definition of API function E2E_P07Forward

Status: DRAFT

Upstream requirements: RS_E2E_08528, RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P07Forward (draft)	

Syntax	`Std_ReturnType E2E_P07Forward (`
	`  E2E_P07ConfigType* ConfigPtr,`
	`  uint32 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P07ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x39	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

▽

△

Description	Protects data which is forwarded by using the E2E profile 07. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft
Available via	E2E.h

⌋

8.3.7.4 E2E_P07Check

[SWS_E2E_00548] Definition of API function E2E_P07Check

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07Check	

Syntax	`Std_ReturnType E2E_P07Check (`
	`  const E2E_P07ConfigType* ConfigPtr,`
	`  E2E_P07CheckStateType* StatePtr,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x78	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 7. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

8.3.7.5 E2E_P07CheckInit

[SWS_E2E_00549] Definition of API function E2E_P07CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07CheckInit	

Syntax	`Std_ReturnType E2E_P07CheckInit (`
	`  E2E_P07CheckStateType* StatePtr`
	`)`	

Service ID [hex]	0x79	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00552]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P07CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF'FF'FF
2. Status to E2E_P07STATUS_ERROR.

⌋

8.3.7.6 E2E_P07MapStatusToSM

[SWS_E2E_00550] Definition of API function E2E_P07MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07MapStatusToSM

Syntax	`E2E_PCheckStatusType E2E_P07MapStatusToSM (`
	`  E2E_PCheckStatusType return,`
	`  E2E_P07CheckStatusType Status`
	`)`

Service ID [hex]	0x80

▽

△

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

⌋

[SWS_E2E_00553]

Upstream requirements: RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P07MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00607]⌋

[SWS_E2E_00607] Mapping of profile 07

Upstream requirements: RS_E2E_08548

⌈

Status	Return value
E2E_P07STATUS_OK or E2E_P07STATUS_OKSOMELOST	E2E_P_OK
E2E_P07STATUS_ERROR	E2E_P_ERROR
E2E_P07STATUS_REPEATED	E2E_P_REPEATED
E2E_P07STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P07STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_00554]

Upstream requirements: RS_E2E_08548

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

8.3.7.7 E2E_P07GetHeaderInfo

[SWS_E2E_91116] Definition of API function E2E_P07GetHeaderInfo ⌈

Service Name	E2E_P07GetHeaderInfo	

Syntax	`Std_ReturnType E2E_P07GetHeaderInfo (`
	`  const E2E_P07ConfigType* ConfigPtr,`
	`  const uint8* DataPtr,`
	`  uint32 Length,`
	`  E2E_P07HeaderInformationType* HeaderInfoPtr`
	`)`	

Service ID [hex]	0x8b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 07. This includes the SequenceCounter	
Available via	E2E.h	

⌋

8.3.8 E2E Profile 7m routines

8.3.8.1 E2E_P07mProtect

[SWS_E2E_91014] Definition of API function E2E_P07mProtect

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07mProtect	

Syntax	`Std_ReturnType E2E_P07mProtect (`
	`  const E2E_P07mConfigType* ConfigPtr,`
	`  E2E_P07mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x4b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.

▽

△

	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID.	
Available via	E2E.h	

⌋

8.3.8.2 E2E_P07mProtectInit

[SWS_E2E_91015] Definition of API function E2E_P07mProtectInit

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07mProtectInit	

Syntax	`Std_ReturnType E2E_P07mProtectInit (`
	`  E2E_P07mProtectStateType* StatePtr`
	`)`	

Service ID [hex]	0x4c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_00596]

Upstream requirements: RS_E2E_08539

⌈In case State is NULL, E2E_P07mProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.8.3 E2E_P07mForward

[SWS_E2E_91016] Definition of API function E2E_P07mForward

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P07mForward (draft)	

Syntax	`Std_ReturnType E2E_P07mForward (`
	`  const E2E_P07mConfigType* ConfigPtr,`
	`  uint32 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P07mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x4d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.8.4 E2E_P07mSourceCheck

[SWS_E2E_91018] Definition of API function E2E_P07mSourceCheck

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07mSourceCheck	

Syntax	`Std_ReturnType E2E_P07mSourceCheck (`
	`  const E2E_P07mConfigType* ConfigPtr,`
	`  E2E_P07mCheckStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x4f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).	
Available via	E2E.h	

⌋

8.3.8.5 E2E_P07mSinkCheck

[SWS_E2E_91017] Definition of API function E2E_P07mSinkCheck

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07mSinkCheck	

Syntax	`Std_ReturnType E2E_P07mSinkCheck (`
	`  const E2E_P07mConfigType* ConfigPtr,`
	`  E2E_P07mCheckStateType* StatePtr,`
	`  uint32* SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType MessageResult,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x4e	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	MessageType	Type of the message (request/response)
	MessageResult	Result of the message (OK/ERROR)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	SourceID	A system-unique identifier of the Data Source.

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 7m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).	
Available via	E2E.h	

⌋

8.3.8.6 E2E_P07mCheckInit

[SWS_E2E_91012] Definition of API function E2E_P07mCheckInit

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07mCheckInit	

Syntax	`Std_ReturnType E2E_P07mCheckInit (`
	`  E2E_P07mCheckStateType* StatePtr`
	`)`	

Service ID [hex]	0x49	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00597]

Upstream requirements: RS_E2E_08539

⌈In case State is NULL, E2E_P07mCheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF'FF'FF.
2. Status to E2E_P07MSTATUS_ERROR.

⌋

8.3.8.7 E2E_P07mMapStatusToSM

[SWS_E2E_91013] Definition of API function E2E_P07mMapStatusToSM

Upstream requirements: RS_E2E_08539, RS_E2E_08527

⌈

Service Name	E2E_P07mMapStatusToSM

Syntax	`E2E_PCheckStatusType E2E_P07mMapStatusToSM (`
	`  Std_ReturnType CheckReturn,`
	`  E2E_P07mCheckStatusType Status`
	`)`

Service ID [hex]	0x4a
Sync/Async	Synchronous

▽

△

Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P07mSinkCheck/E2E_P07mSource Check function
	Status	Status determined by E2E_P07mSinkCheck/E2E_P07mSource Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 7m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 7m delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_00598]

Upstream requirements: RS_E2E_08539

⌈If CheckReturn = E2E_E_OK, then the function E2E_P07mMapStatusToSMshall return the values depending on the value of Status: See [SWS_E2E_00611]⌋

[SWS_E2E_00611] Mapping of profile 07m

Upstream requirements: RS_E2E_08539

⌈

Status	Return value
E2E_P07MSTATUS_OK or E2E_P07MSTATUS_ OKSOMELOST	E2E_P_OK
E2E_P07MSTATUS_ERROR	E2E_P_ERROR
E2E_P07MSTATUS_REPEATED	E2E_P_REPEATED
E2E_P07MSTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P07MSTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_00599]

Upstream requirements: RS_E2E_08539

⌈If CheckReturn != E2E_E_OK, then the function E2E_P07mMapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.8.8 E2E_P07mGetHeaderInfo

[SWS_E2E_91120] Definition of API function E2E_P07mGetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P07mGetHeaderInfo	

Syntax	`Std_ReturnType E2E_P07mGetHeaderInfo (`
	`  const E2E_P07mConfigType* ConfigPtr,`
	`  const uint8* DataPtr,`
	`  uint32 Length,`
	`  E2E_P07mHeaderInformationType* HeaderInfoPtr`
	`)`	

Service ID [hex]	0x8f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter and Source ID) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 07m. This includes SequenceCounter and SourceID.	
Available via	E2E.h	

⌋

8.3.9 E2E Profile 8 routines

8.3.9.1 E2E_P08Protect

[SWS_E2E_91036] Definition of API function E2E_P08Protect

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P08Protect	

Syntax	`Std_ReturnType E2E_P08Protect (`
	`  const E2E_P08ConfigType* ConfigPtr,`
	`  E2E_P08ProtectStateType* StatePtr,`
	`  uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x57	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.

▽

△

	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	DataPtr	Pointer to Data to be transmitted.

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 08. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

⌋

8.3.9.2 E2E_P08ProtectInit

[SWS_E2E_91037] Definition of API function E2E_P08ProtectInit

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P08ProtectInit	

Syntax	`Std_ReturnType E2E_P08ProtectInit (`
	`  E2E_P08ProtectStateType* StatePtr`
	`)`	

Service ID [hex]	0x58	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Initializes the protection state.	
Available via	E2E.h	

⌋

RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_10004]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P08ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.9.3 E2E_P08Forward

[SWS_E2E_91038] Definition of API function E2E_P08Forward

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P08Forward (draft)	

Syntax	`Std_ReturnType E2E_P08Forward (`
	`  const E2E_P08ConfigType* ConfigPtr,`
	`  uint32 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P08ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x59	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 08. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.9.4 E2E_P08Check

[SWS_E2E_91039] Definition of API function E2E_P08Check

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P08Check

Syntax	`Std_ReturnType E2E_P08Check (`
	`  const E2E_P08ConfigType* ConfigPtr,`
	`  E2E_P08CheckStateType* StatePtr,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`

Service ID [hex]	0x5a

▽

△

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 08. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

(RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.9.5 E2E_P08CheckInit

[SWS_E2E_91040] Definition of API function E2E_P08CheckInit

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P08CheckInit	

Syntax	`Std_ReturnType E2E_P08CheckInit (`
	`  E2E_P08CheckStateType* StatePtr`
	`)`	

Service ID [hex]	0x5b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_10005]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P08CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF’FF’FF’FF

2. Status to E2E_P08STATUS_ERROR.

⌋

8.3.9.6 E2E_P08MapStatusToSM

[SWS_E2E_91041] Definition of API function E2E_P08MapStatusToSM

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P08MapStatusToSM	
Syntax	`E2E_PCheckStatusType E2E_P08MapStatusToSM ( Std_ReturnType CheckReturn, E2E_P08CheckStatusType Status )`	
Service ID [hex]	0x5C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P08Check function
	Status	Status determined by E2E_P08Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 08 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 08 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_10006]

Upstream requirements: RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P08MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00613]⌋

[SWS_E2E_00613] Mapping of profile 08

Upstream requirements: RS_E2E_08548

⌈

Status	Return value
E2E_P08STATUS_OK or E2E_P08STATUS_OKSOMELOST	E2E_P_OK
E2E_P08STATUS_ERROR	E2E_P_ERROR
E2E_P08STATUS_REPEATED	E2E_P_REPEATED
E2E_P08STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P08STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_10007]

Upstream requirements: RS_E2E_08548

⌈If CheckReturn != E2E_E_OK, then the function E2E_P08MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.9.7 E2E_P08GetHeaderInfo

[SWS_E2E_91117] Definition of API function E2E_P08GetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P08GetHeaderInfo	
Syntax	`Std_ReturnType E2E_P08GetHeaderInfo ( const E2E_P08ConfigType* ConfigPtr, const uint8* DataPtr, uint32 Length, E2E_P08HeaderInformationType* HeaderInfoPtr )`	
Service ID [hex]	0x8c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 08. This includes the SequenceCounter	
Available via	E2E.h	

⌋

8.3.10 E2E Profile 8m routines

8.3.10.1 E2E_P08mProtect

[SWS_E2E_91081] Definition of API function E2E_P08mProtect

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mProtect	

Syntax	`Std_ReturnType E2E_P08mProtect (`
	`  const E2E_P08mConfigType* ConfigPtr,`
	`  E2E_P08mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x60	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 08m. This includes CRC calculation, handling of counter, Data ID, Message Type, Message Result and Source ID.	
Available via	E2E.h	

⌋

8.3.10.2 E2E_P08mProtectInit

[SWS_E2E_91082] Definition of API function E2E_P08mProtectInit

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mProtectInit

Syntax	`Std_ReturnType E2E_P08mProtectInit (`
	`  E2E_P08mProtectStateType* StatePtr`
	`)`

▽

△

Service ID [hex]	0x61	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

RS_E2E_08527, RS_E2E_08539)

[SWS_E2E_91053]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P08mProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.10.3 E2E_P08mForward

[SWS_E2E_91083] Definition of API function E2E_P08mForward

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mForward (draft)	

Syntax	`Std_ReturnType E2E_P08mForward (`
	`  const E2E_P08mConfigType* ConfigPtr,`
	`  uint32 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P08mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x62	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
	SourceID	A system-unique identifier of the Data Source.

▽

△

	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 08m. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.10.4 E2E_P08mSourceCheck

[SWS_E2E_91084] Definition of API function E2E_P08mSourceCheck

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mSourceCheck	

Syntax	`Std_ReturnType E2E_P08mSourceCheck (`
	`  const E2E_P08mConfigType* ConfigPtr,`
	`  E2E_P08mCheckStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x63	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

▽

△

Description	Checks the Data received using the E2E profile 8m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
Available via	E2E.h

⌋

8.3.10.5 E2E_P08mSinkCheck

[SWS_E2E_91085] Definition of API function E2E_P08mSinkCheck

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mSinkCheck	

Syntax	`Std_ReturnType E2E_P08mSinkCheck (`
	`  const E2E_P08mConfigType* ConfigPtr,`
	`  E2E_P08mCheckStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x64	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	SourceID	A system-unique identifier of the Data Source.

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 8m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).	
Available via	E2E.h	

⌋

8.3.10.6 E2E_P08mCheckInit

[SWS_E2E_91086] Definition of API function E2E_P08mCheckInit

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mCheckInit	
Syntax	`Std_ReturnType E2E_P08mCheckInit ( E2E_P08mCheckStateType* StatePtr )`	
Service ID [hex]	0x65	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK` `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_91072]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P08mCheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0x FF'FF'FF'FF.
2. Status to E2E_P08MSTATUS_ERROR.⌋

8.3.10.7 E2E_P08mMapStatusToSM

[SWS_E2E_91087] Definition of API function E2E_P08mMapStatusToSM

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P08mMapStatusToSM
Syntax	`E2E_PCheckStatusType E2E_P08mMapStatusToSM ( Std_ReturnType CheckReturn, E2E_P08mCheckStatusType Status )`
Service ID [hex]	0x66
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	CheckReturn	Return value of the E2E_P08mSinkCheck/E2E_P08mSource Check function
	Status	Status determined by E2E_P08mSinkCheck/E2E_P08mSource Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 08m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 08m delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_91059]

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈If CheckReturn = E2E_E_OK, then the function E2E_P08mMapStatusToSMshall return the values depending on the value of Status: See [SWS_E2E_00614]⌋

[SWS_E2E_00614] Mapping of profile 08m

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Status	Return value
E2E_P08MSTATUS_OK or E2E_P08MSTATUS_ OKSOMELOST	E2E_P_OK
E2E_P08MSTATUS_ERROR	E2E_P_ERROR
E2E_P08MSTATUS_REPEATED	E2E_P_REPEATED
E2E_P08MSTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P08MSTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_91060]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn != E2E_E_OK, then the function E2E_P08mMapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.10.8 E2E_P08mGetHeaderInfo

[SWS_E2E_91121] Definition of API function E2E_P08mGetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P08mGetHeaderInfo	

Syntax	`Std_ReturnType E2E_P08mGetHeaderInfo (`
	`  const E2E_P08mConfigType* ConfigPtr,`
	`  const uint8* DataPtr,`
	`  uint32 Length,`
	`  E2E_P08mHeaderInformationType* HeaderInfoPtr`
	`)`	

Service ID [hex]	0x90	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter and Source ID) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 08m. This includes SequenceCounter and SourceID.	
Available via	E2E.h	

⌋

8.3.11 E2E Profile 11 routines

8.3.11.1 E2E_P11Protect

[SWS_E2E_00575] Definition of API function E2E_P11Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P11Protect	

Syntax	`Std_ReturnType E2E_P11Protect (`
	`  const E2E_P11ConfigType* ConfigPtr,`
	`  E2E_P11ProtectStateType* StatePtr,`
	`  uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x3b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.

▽

△

	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`, `E2E_E_OK`, `E2E_E_INPUTERR_WRONG`, For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 11. This includes checksum calculation, handling of counter.	
Available via	E2E.h	

⌋

8.3.11.2 E2E_P11ProtectInit

[SWS_E2E_00576] Definition of API function E2E_P11ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

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

⌋

[SWS_E2E_00555]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P11ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.11.3 E2E_P11Forward

[SWS_E2E_00587] Definition of API function E2E_P11Forward

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P11Forward (draft)	

Syntax	`Std_ReturnType E2E_P11Forward (`
	`  const E2E_P11ConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P11ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x36	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 11. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.11.4 E2E_P11Check

[SWS_E2E_00572] Definition of API function E2E_P11Check

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P11Check

Syntax	`Std_ReturnType E2E_P11Check (`
	`  const E2E_P11ConfigType* ConfigPtr,`
	`  E2E_P11CheckStateType* StatePtr,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`

Service ID [hex]	0x81

▽

△

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`, `E2E_E_OK`, `E2E_E_INPUTERR_WRONG`, For definitions for return values, see SWS_E2E_00047.
Description	Checks the Data received using the E2E profile 11. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

8.3.11.5 E2E_P11CheckInit

[SWS_E2E_00573] Definition of API function E2E_P11CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P11CheckInit	
Syntax	`Std_ReturnType E2E_P11CheckInit ( E2E_P11CheckStateType* StatePtr )`	
Service ID [hex]	0x82	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`.
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_00556]

Upstream requirements: RS_E2E_08528

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

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

⌋

8.3.11.6 E2E_P11MapStatusToSM

[SWS_E2E_00574] Definition of API function E2E_P11MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P11MapStatusToSM	
Syntax	`E2E_PCheckStatusType E2E_P11MapStatusToSM ( Std_ReturnType CheckReturn, E2E_P11CheckStatusType Status )`	
Service ID [hex]	0x3a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P11Check function
	Status	Status determined by E2E_P11Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 11 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 11 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_00557]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P11MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00608]⌋

[SWS_E2E_00608] Mapping of profile 11

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈

Status	Return value
E2E_P11STATUS_OK or E2E_P11STATUS_OKSOMELOST	E2E_P_OK
E2E_P11STATUS_ERROR	E2E_P_ERROR
E2E_P11STATUS_REPEATED	E2E_P_REPEATED
E2E_P11STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P11STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_00558]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

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

8.3.12 E2E Profile 22 routines

8.3.12.1 E2E_P22Protect

[SWS_E2E_00580] Definition of API function E2E_P22Protect

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P22Protect	

Syntax	`Std_ReturnType E2E_P22Protect (`
	`  const E2E_P22ConfigType* ConfigPtr,`
	`  E2E_P22ProtectStateType* StatePtr,`
	`  uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x40	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`, `E2E_E_OK`, `E2E_E_INPUTERR_WRONG`, For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 22. This includes checksum calculation, handling of counter.	
Available via	E2E.h	

⌋

8.3.12.2 E2E_P22ProtectInit

[SWS_E2E_00581] Definition of API function E2E_P22ProtectInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P22ProtectInit	
Syntax	`Std_ReturnType E2E_P22ProtectInit ( E2E_P22ProtectStateType* StatePtr )`	
Service ID [hex]	0x41	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed, `E2E_E_OK`.
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_00559]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P22ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.12.3 E2E_P22Forward

[SWS_E2E_00589] Definition of API function E2E_P22Forward

Status: DRAFT

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P22Forward (draft)	
Syntax	`Std_ReturnType E2E_P22Forward ( E2E_P22ConfigType* ConfigPtr, uint16 Length, E2E_PCheckStatusType ForwardStatus, E2E_P22ProtectStateType* StatePtr, uint8* DataPtr )`	
Service ID [hex]	0x37	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.

▽

△

	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 22. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.12.4 E2E_P22Check

[SWS_E2E_00577] Definition of API function E2E_P22Check

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P22Check	

Syntax	`Std_ReturnType E2E_P22Check (`
	`  const E2E_P22ConfigType* ConfigPtr,`
	`  E2E_P22CheckStateType* StatePtr,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x3d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`, `E2E_E_OK`, `E2E_E_INPUTERR_WRONG`, For definitions for return values, see SWS_E2E_00047.
Description	Checks the Data received using the E2E profile 22. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

8.3.12.5 E2E_P22CheckInit

[SWS_E2E_00578] Definition of API function E2E_P22CheckInit

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

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

⌋

[SWS_E2E_00560]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P22CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xF
2. Status to E2E_P22STATUS_ERROR.

⌋

8.3.12.6 E2E_P22MapStatusToSM

[SWS_E2E_00579] Definition of API function E2E_P22MapStatusToSM

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P22MapStatusToSM
Syntax	`E2E_PCheckStatusType E2E_P22MapStatusToSM ( Std_ReturnType CheckReturn, E2E_P22CheckStatusType Status )`
Service ID [hex]	0x3f
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	CheckReturn	Return value of the E2E_P22Check function
	Status	Status determined by E2E_P22Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 22 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 22 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_00561]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P22MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00609]⌋

[SWS_E2E_00609] Mapping of profile 22

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈

Status	Return value
E2E_P22STATUS_OK or E2E_P22STATUS_OKSOMELOST	E2E_P_OK
E2E_P22STATUS_ERROR	E2E_P_ERROR
E2E_P22STATUS_REPEATED	E2E_P_REPEATED
E2E_P22STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P22STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_00562]

Upstream requirements: RS_E2E_08528, RS_E2E_08548

⌈If CheckReturn != E2E_E_OK, then the function E2E_P22MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.13 E2E Profile 44 routines

8.3.13.1 E2E_P44Protect

[SWS_E2E_91027] Definition of API function E2E_P44Protect

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44Protect	

Syntax	`Std_ReturnType E2E_P44Protect (`
	`  const E2E_P44ConfigType* ConfigPtr,`
	`  E2E_P44ProtectStateType* StatePtr,`
	`  uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x50	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 44. This includes checksum calculation, handling of counter and Data ID.	
Available via	E2E.h	

⌋

8.3.13.2 E2E_P44ProtectInit

[SWS_E2E_91028] Definition of API function E2E_P44ProtectInit

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44ProtectInit

Syntax	`Std_ReturnType E2E_P44ProtectInit (`
	`  E2E_P44ProtectStateType* StatePtr`
	`)`

Service ID [hex]	0x51
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_10001]

Upstream requirements: RS_E2E_08539

⌈In case State is NULL, E2E_P44ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.13.3 E2E_P44Forward

[SWS_E2E_91029] Definition of API function E2E_P44Forward

Status: DRAFT
Upstream requirements: RS_E2E_08527

⌈

Service Name	E2E_P44Forward (draft)	

Syntax	`Std_ReturnType E2E_P44Forward (`
	`  const E2E_P44ConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_P44CheckStatusType ForwardStatus,`
	`  E2E_P44ProtectStateType* StatePtr,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x52	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 44. This includes checksum calculation, handling of counter and Data ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.13.4 E2E_P44Check

[SWS_E2E_91030] Definition of API function E2E_P44Check

Upstream requirements: RS_E2E_08527

⌈

Service Name	E2E_P44Check	
Syntax	`Std_ReturnType E2E_P44Check ( const E2E_P44ConfigType* ConfigPtr, E2E_P44CheckStateType* StatePtr, const uint8* DataPtr, uint16 Length )`	
Service ID [hex]	0x53	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 44. This includes CRC calculation, handling of Counter and Data ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link.	
Available via	E2E.h	

⌋

8.3.13.5 E2E_P44CheckInit

[SWS_E2E_91031] Definition of API function E2E_P44CheckInit

Upstream requirements: RS_E2E_08527

⌈

Service Name	E2E_P44CheckInit	
Syntax	`Std_ReturnType E2E_P44CheckInit ( E2E_P44CheckStateType* StatePtr )`	
Service ID [hex]	0x55	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`

▽

△

Description	Initializes the check state.
Available via	E2E.h

⌋

[SWS_E2E_10002]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P44CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0xFF'FF.
2. Status to E2E_P44STATUS_ERROR.

⌋

8.3.13.6 E2E_P44MapStatusToSM

[SWS_E2E_91032] Definition of API function E2E_P44MapStatusToSM

Upstream requirements: RS_E2E_08527

⌈

Service Name	E2E_P44MapStatusToSM	
Syntax	`E2E_PCheckStatusType E2E_P44MapStatusToSM ( Std_ReturnType CheckReturn, E2E_P44CheckStatusType Status )`	
Service ID [hex]	0x56	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P44Check function
	Status	Status determined by E2E_P44Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 44 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 44 delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_10003]

Upstream requirements: RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P44MapStatusToSMshall return the values depending on the value of Status: See [SWS_E2E_00612] If CheckReturn

!= E2E_E_OK, then the function E2E_P44MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

[SWS_E2E_00612] Mapping of profile 44

Upstream requirements: RS_E2E_08548

⌈

Status	Return value
E2E_P44STATUS_OK or E2E_P44STATUS_OKSOMELOST	E2E_P_OK
E2E_P44STATUS_ERROR	E2E_P_ERROR
E2E_P44STATUS_REPEATED	E2E_P_REPEATED
E2E_P44STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P44STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

8.3.13.7 E2E_P44GetHeaderInfo

[SWS_E2E_91118] Definition of API function E2E_P44GetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P44GetHeaderInfo	
Syntax	`Std_ReturnType E2E_P44GetHeaderInfo ( const E2E_P44ConfigType* ConfigPtr, const uint8* DataPtr, uint16 Length, E2E_P44HeaderInformationType* HeaderInfoPtr )`	
Service ID [hex]	0x8d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 44. This includes the SequenceCounter	
Available via	E2E.h	

⌋

8.3.14 E2E Profile 44m routines

8.3.14.1 E2E_P44mProtect

[SWS_E2E_91088] Definition of API function E2E_P44mProtect

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mProtect	

Syntax	`Std_ReturnType E2E_P44mProtect (`
	`  const E2E_P44mConfigType* ConfigPtr,`
	`  E2E_P44mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x68	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects the array/buffer to be transmitted using the E2E profile 44m. This includes CRC calculation, handling of counter, Data ID, Message Type, Message Result and Source ID.	
Available via	E2E.h	

⌋

8.3.14.2 E2E_P44mProtectInit

[SWS_E2E_91089] Definition of API function E2E_P44mProtectInit

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mProtectInit

Syntax	`Std_ReturnType E2E_P44mProtectInit (`
	`  E2E_P44mProtectStateType* StatePtr`
	`)`

▽

△

Service ID [hex]	0x69	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the protection state.	
Available via	E2E.h	

⌋

[SWS_E2E_91063]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P44mProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.14.3 E2E_P44mForward

[SWS_E2E_91090] Definition of API function E2E_P44mForward

Status: DRAFT

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mForward (draft)	

Syntax	`Std_ReturnType E2E_P44mForward (`
	`  const E2E_P44mConfigType* ConfigPtr,`
	`  uint16 Length,`
	`  E2E_PCheckStatusType ForwardStatus,`
	`  E2E_P44mProtectStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  uint8* DataPtr`
	`)`	

Service ID [hex]	0x6A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes.
	ForwardStatus	E2E Status of the received message
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)

▽

△

Parameters (inout)	StatePtr	Pointer to port/data communication state.
	DataPtr	Pointer to Data to be transmitted.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Protects data which is forwarded by using the E2E profile 44m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. Detected Errors of received message will be reconstruct on output data. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.14.4 E2E_P44mSourceCheck

[SWS_E2E_91091] Definition of API function E2E_P44mSourceCheck

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mSourceCheck	

Syntax	`Std_ReturnType E2E_P44mSourceCheck (`
	`  const E2E_P44mConfigType* ConfigPtr,`
	`  E2E_P44mCheckStateType* StatePtr,`
	`  uint32 SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x6B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	SourceID	A system-unique identifier of the Data Source.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

▽

△

Description	Checks the Data received using the E2E profile 44m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data source (i.e., in case of C/S communication at the client).
Available via	E2E.h

⌋

8.3.14.5 E2E_P44mSinkCheck

[SWS_E2E_91092] Definition of API function E2E_P44mSinkCheck

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mSinkCheck	

Syntax	`Std_ReturnType E2E_P44mSinkCheck (`
	`  const E2E_P44mConfigType* ConfigPtr,`
	`  E2E_P44mCheckStateType* StatePtr,`
	`  uint32* SourceID,`
	`  Std_MessageTypeType MessageType,`
	`  Std_MessageResultType Messageresult,`
	`  const uint8* DataPtr,`
	`  uint16 Length`
	`)`	

Service ID [hex]	0x6C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	MessageType	Type of the message (request/response)
	Messageresult	Result of the message (OK/Error)
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	SourceID	A system-unique identifier of the Data Source.

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 44m. This includes CRC calculation, handling of Counter, Data ID, Message Type, Message Result, and Source ID. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. This function is intended for usage at the data sink (i.e., in case of C/S communication at the server).	
Available via	E2E.h	

⌋

8.3.14.6 E2E_P44mCheckInit

[SWS_E2E_91093] Definition of API function E2E_P44mCheckInit

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mCheckInit	
Syntax	`Std_ReturnType E2E_P44mCheckInit ( E2E_P44mCheckStateType* StatePtr )`	
Service ID [hex]	0x6E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK` `E2E_E_OK`
Description	Initializes the check state	
Available via	E2E.h	

⌋

[SWS_E2E_91068]

Upstream requirements: RS_E2E_08528

⌈In case State is NULL, E2E_P44mCheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0x FF'FF.
2. Status to E2E_P44MSTATUS_ERROR.⌋

8.3.14.7 E2E_P44mMapStatusToSM

[SWS_E2E_91094] Definition of API function E2E_P44mMapStatusToSM

Upstream requirements: RS_E2E_08527, RS_E2E_08528

⌈

Service Name	E2E_P44mMapStatusToSM	
Syntax	`E2E_PCheckStatusType E2E_P44mMapStatusToSM ( Std_ReturnType CheckReturn, E2E_P44mCheckStatusType Status )`	
Service ID [hex]	0x6F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P44mCheck function

▽

△

	Status	Status determined by E2E_P44mCheck function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 44m to a generic check status, which can be used by E2E state machine check function. The E2E Profile 44m delivers a more fine-granular status, but this is not relevant for the E2E state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_91070]

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈If CheckReturn = E2E_E_OK, then the function E2E_P44mMapStatusToSMshall return the values depending on the value of Status: See [SWS_E2E_00615]⌋

[SWS_E2E_00615] Mapping of profile 44m

Upstream requirements: RS_E2E_08528, RS_E2E_08539, RS_E2E_08527

⌈

Status	Return value
E2E_P44MSTATUS_OK or E2E_P44MSTATUS_OKSOMELOST	E2E_P_OK
E2E_P44MSTATUS_ERROR	E2E_P_ERROR
E2E_P44MSTATUS_REPEATED	E2E_P_REPEATED
E2E_P44MSTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P44MSTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_91071]

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈If CheckReturn != E2E_E_OK, then the function E2E_P44mMapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.14.8 E2E_P44mGetHeaderInfo

[SWS_E2E_91122] Definition of API function E2E_P44mGetHeaderInfo

Upstream requirements: RS_E2E_08528, RS_E2E_08527

⌈

Service Name	E2E_P44mGetHeaderInfo	

Syntax	`Std_ReturnType E2E_P44mGetHeaderInfo (`
	`  const E2E_P44mConfigType* ConfigPtr,`
	`  const uint8* DataPtr,`
	`  uint16 Length,`
	`  E2E_P44mHeaderInformationType* HeaderInfoPtr`
	`)`	

Service ID [hex]	0x91	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data
	Length	Length of the data in bytes
Parameters (inout)	None	
Parameters (out)	HeaderInfoPtr	Profile specific header elements (SequenceCounter and Source ID) retrieved from DataPtr
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`: Length or DataPtr NULL `E2E_E_OK`: Retrieval successful
Description	Retrieves header element data for further processing when using the E2E profile 44m. This includes SequenceCounter and SourceID.	
Available via	E2E.h	

⌋

8.3.15 E2E Profile 76 routines

8.3.15.1 E2E_P76Protect

[SWS_E2E_91099] Definition of API function E2E_P76Protect

Status: DRAFT

Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P76Protect (draft)

Syntax	`Std_ReturnType E2E_P76Protect (`
	`  const E2E_P76ConfigType* ConfigPtr,`
	`  E2E_P76ProtectStateType* StatePtr,`
	`  uint8* DataPtr,`
	`  uint32 Length`
	`)`

Service ID [hex]	0x87
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	ConfigPtr	Pointer to static configuration.
	Length	Length of the data in bytes
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	DataPtr	Pointer to Data to be transmitted.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Protects the array/buffer to be transmitted using the E2E profile 76. This includes checksum calculation, handling of counter. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

8.3.15.2 E2E_P76ProtectInit

[SWS_E2E_91100] Definition of API function E2E_P76ProtectInit

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P76ProtectInit (draft)	

Syntax	`Std_ReturnType E2E_P76ProtectInit (`
	`  E2E_P76ProtectStateType* StatePtr`
	`)`	

Service ID [hex]	0x86	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	E2E_E_INPUTERR_NULL E2E_E_INPUTERR_WRONG E2E_E_OK For definitions for return values, see SWS_E2E_00047.
Description	Initializes the protection state. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

[SWS_E2E_00624] P76Protect - State NULL

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P76ProtectInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting Counter to 0.⌋

8.3.15.3 E2E_P76Check

[SWS_E2E_91102] Definition of API function E2E_P76Check

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P76Check (draft)	

Syntax	`Std_ReturnType E2E_P76Check (`
	`  const E2E_P76ConfigType* ConfigPtr,`
	`  E2E_P76CheckStateType* StatePtr,`
	`  const uint8* DataPtr,`
	`  uint32 Length`
	`)`	

Service ID [hex]	0x84	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to static configuration.
	DataPtr	Pointer to received data.
	Length	Length of the data in bytes.
Parameters (inout)	StatePtr	Pointer to received data.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the Data received using the E2E profile 76. This includes CRC calculation, handling of Counter. The function checks only one single data in one cycle, it does not determine/compute the accumulated state of the communication link. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

(RS_E2E_08528, RS_E2E_08539, RS_E2E_08527)

8.3.15.4 E2E_P76CheckInit

[SWS_E2E_91101] Definition of API function E2E_P76CheckInit

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P76CheckInit (draft)	
Syntax	`Std_ReturnType E2E_P76CheckInit ( E2E_P76CheckStateType* StatePtr )`	
Service ID [hex]	0x83	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the check state **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

[SWS_E2E_00625] P76Check - State NULL

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In case State is NULL, E2E_P76CheckInit shall return immediately with E2E_E_INPUTERR_NULL. Otherwise, it shall initialize the state structure, setting:

1. Counter to 0x1F
2. Status to E2E_P76STATUS_ERROR.

⌋

8.3.15.5 E2E_P76MapStatusToSM

[SWS_E2E_91098] Definition of API function E2E_P76MapStatusToSM

Status: DRAFT
Upstream requirements: RS_E2E_08527, RS_E2E_08539

⌈

Service Name	E2E_P76MapStatusToSM (draft)	
Syntax	`E2E_PCheckStatusType E2E_P76MapStatusToSM ( Std_ReturnType CheckReturn, E2E_P76CheckStatusType Status )`	
Service ID [hex]	0x5d	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CheckReturn	Return value of the E2E_P76Check function
	Status	Status determined by E2E_P76Check function
Parameters (inout)	None	
Parameters (out)	None	
Return value	E2E_PCheckStatusType	Profile-independent status of the reception on one single Data in one cycle.
Description	The function maps the check status of Profile 76 to a generic check status, which can be used by E2E state machine check function. The E2E Profile 76 delivers a more fine-granular status, but this is not relevant for the E2E state machine. **Tags:** atp.Status=draft	
Available via	E2E.h	

⌋

[SWS_E2E_00626] P76 Status Mapping

Status: DRAFT
Upstream requirements: RS_E2E_08548

⌈If CheckReturn = E2E_E_OK, then the function E2E_P76MapStatusToSM shall return the values depending on the value of Status: See [SWS_E2E_00623]⌋

[SWS_E2E_00623] Mapping of profile 76

Status: DRAFT
Upstream requirements: RS_E2E_08548

⌈

Status	Return value
E2E_P76STATUS_OK or E2E_P76STATUS_OKSOMELOST	E2E_P_OK
E2E_P76STATUS_ERROR	E2E_P_ERROR
E2E_P76STATUS_REPEATED	E2E_P_REPEATED
E2E_P76STATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_P76STATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

[SWS_E2E_00627] P76 Error

Status: DRAFT
Upstream requirements: RS_E2E_08548

⌈If CheckReturn != E2E_E_OK, then the function E2E_P76MapStatusToSM() shall return E2E_P_ERROR (regardless of value of Status).⌋

8.3.16 E2E State machine routines

8.3.16.1 E2E_SMCheck

[SWS_E2E_00340] Definition of API function E2E_SMCheck

Upstream requirements: RS_E2E_08548

⌈

Service Name	E2E_SMCheck	
Syntax	`Std_ReturnType E2E_SMCheck ( E2E_PCheckStatusType ProfileStatus, const E2E_SMConfigType* ConfigPtr, E2E_SMCheckStateType* StatePtr )`	
Service ID [hex]	0x30	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ProfileStatus	Profile-independent status of the reception on one single Data in one cycle
	ConfigPtr	Pointer to static configuration.
Parameters (inout)	StatePtr	Pointer to port/data communication state.
Parameters (out)	None	

Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL`
		`E2E_E_INPUTERR_WRONG`
		`E2E_E_OK`
		`E2E_E_WRONGSTATE`
		For definitions for return values, see SWS_E2E_00047.

Description	Checks the communication channel. It determines if the data can be used for safety-related application, based on history of checks performed by a corresponding E2E_P0XCheck() function.	
Available via	E2E.h	

⌋

[SWS_E2E_00371]

Upstream requirements: RS_E2E_08528

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

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

8.3.16.2 E2E_SMCheckInit

[SWS_E2E_00353] Definition of API function E2E_SMCheckInit

Upstream requirements: RS_E2E_08528

⌈

Service Name	E2E_SMCheckInit	
Syntax	`Std_ReturnType E2E_SMCheckInit ( E2E_SMCheckStateType* StatePtr, const E2E_SMConfigType* ConfigPtr )`	
Service ID [hex]	0x31	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration of the state machine
Parameters (inout)	None	
Parameters (out)	StatePtr	Pointer to port/data communication state.
Return value	Std_ReturnType	`E2E_E_INPUTERR_NULL` - null pointer passed `E2E_E_OK`
Description	Initializes the state machine.	
Available via	E2E.h	

⌋

[SWS_E2E_00370]

Upstream requirements: RS_E2E_08528

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

Else (i.e. both pointers arenot NULL), the function E2E_SMCheckInit shall initialize the State structure, setting:

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

and it shall return with E2E_E_OK.⌋

8.4 Scheduled functions

None. The E2E library does not have scheduled functions.

8.5 Expected interfaces

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

[SWS_E2E_00110]

Upstream requirements: RS_E2E_08528

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

8.5.1 Mandatory Interfaces

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

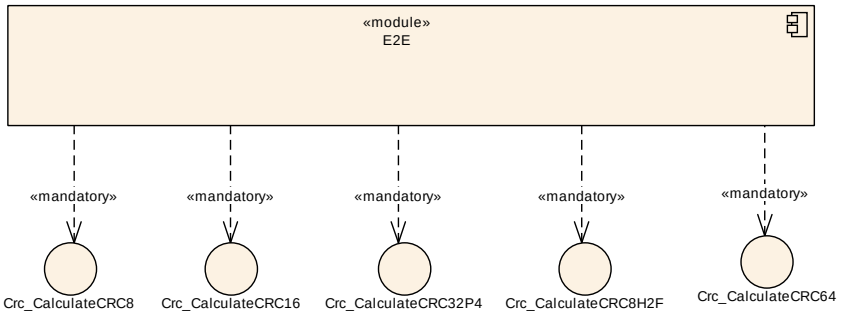


Figure 8.51: Expected mandatory interfaces by E2E library

9 Sequence diagrams

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

9.1 Sender

[UC_E2E_00202]

Upstream requirements: RS_E2E_08528

⌈During its initialization, the Sender shall instantiate the structures PXXConfigType and PXXProtectStateType, separately for each Data to be protected.⌋

[UC_E2E_00203]

Upstream requirements: RS_E2E_08528

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

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

[UC_E2E_00204]

Upstream requirements: RS_E2E_08528

⌈During its initialization, the Sender shall initialize the E2E_PXXProtectStateType for each Data, with the configured following values: Counter = 0.⌋

[UC_E2E_00205]

Upstream requirements: RS_E2E_08528

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

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

9.1.1 Sender of data elements

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

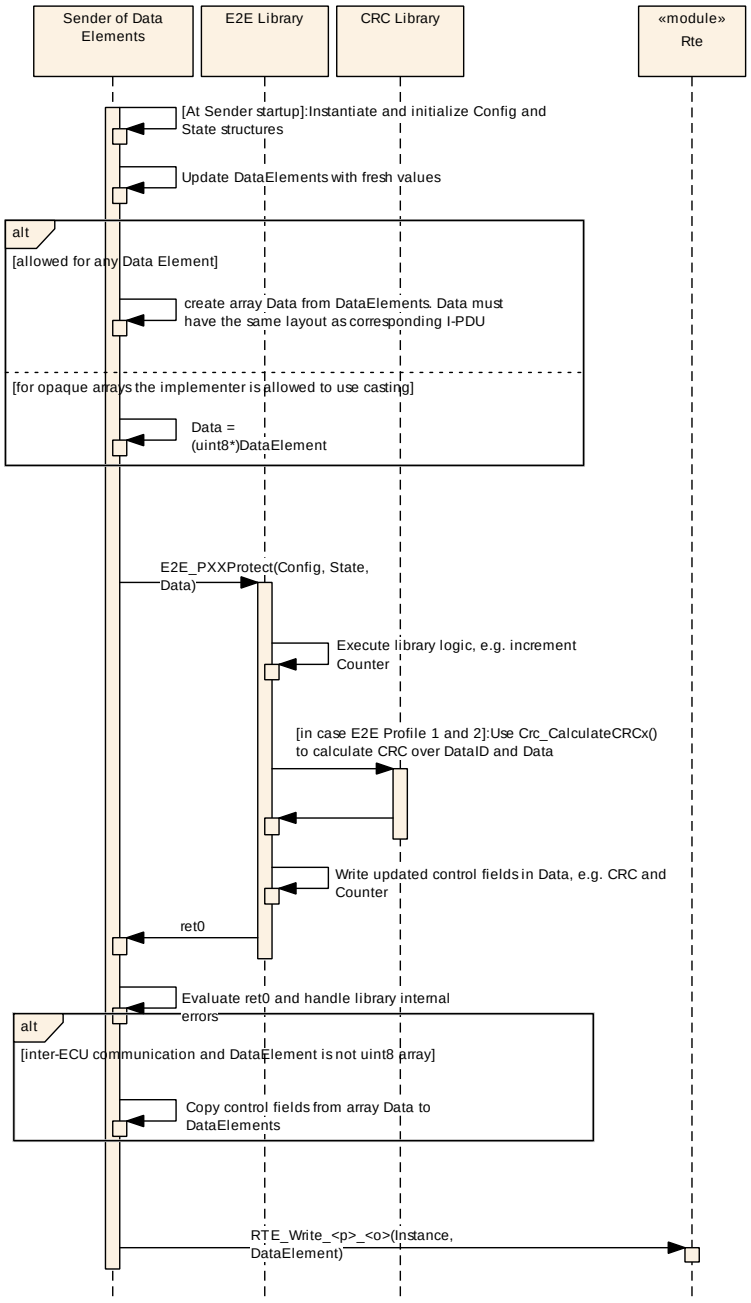


Figure 9.1: Sender of data elements

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

[UC_E2E_00230]

Upstream requirements: RS_E2E_08528

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

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

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

[UC_E2E_00232]

Upstream requirements: RS_E2E_08528

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

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

The offsets of control fields in Data are defined in Software Component Template meta-class EndToEndDescription.

9.1.2 Sender at signal group level

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

The diagram shows the example when there is only one E2E-protected signal group in the I-PDU, but in general it is possible to have several of them (0 or 1 E2E-protections per signal group). In such case, the sender of I-PDUs invokes E2E_PXXProtect on each E2E-protected signal group.

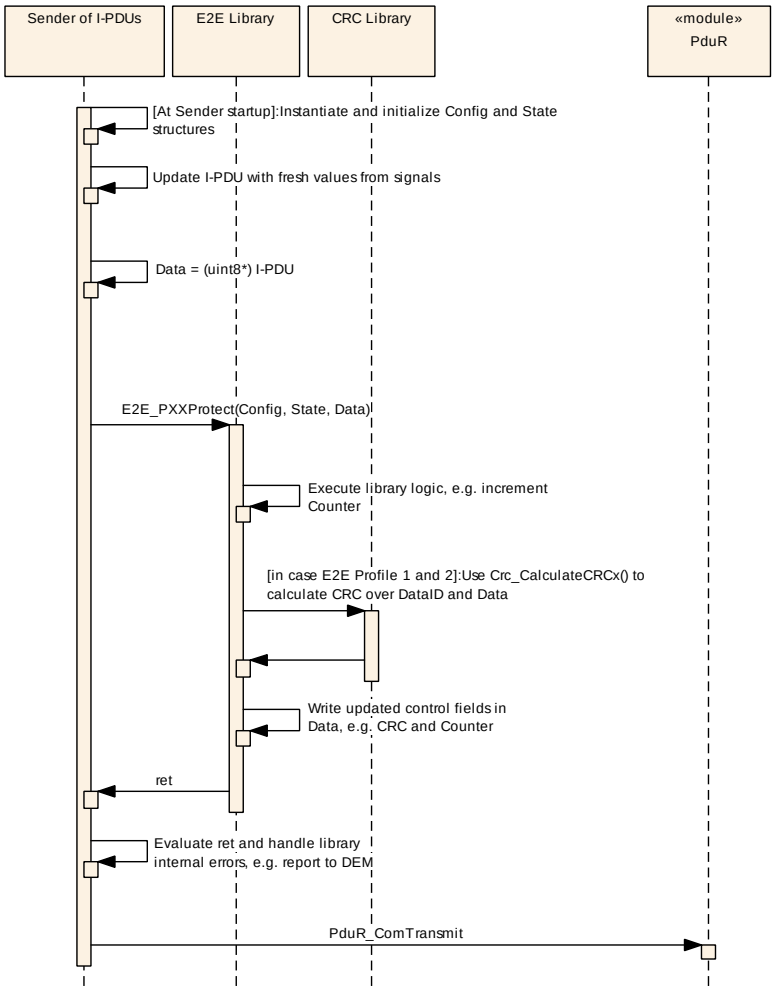


Figure 9.2: Sender of I-PDUs

9.2 Receiver

[UC_E2E_00206]

Upstream requirements: RS_E2E_08528

⌈During its initialization, the Receiver shall instantiate the structures PXXConfigType and PXXReceiverType.⌋

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

State -> SyncCounter := 0;

Config -> MaxNoNewOrRepeatedData := 14 (when using Profile 1);

Config -> MaxNoNewOrRepeatedData := 15 (when using Profile 2);

Config -> SyncCounterInit := 0;

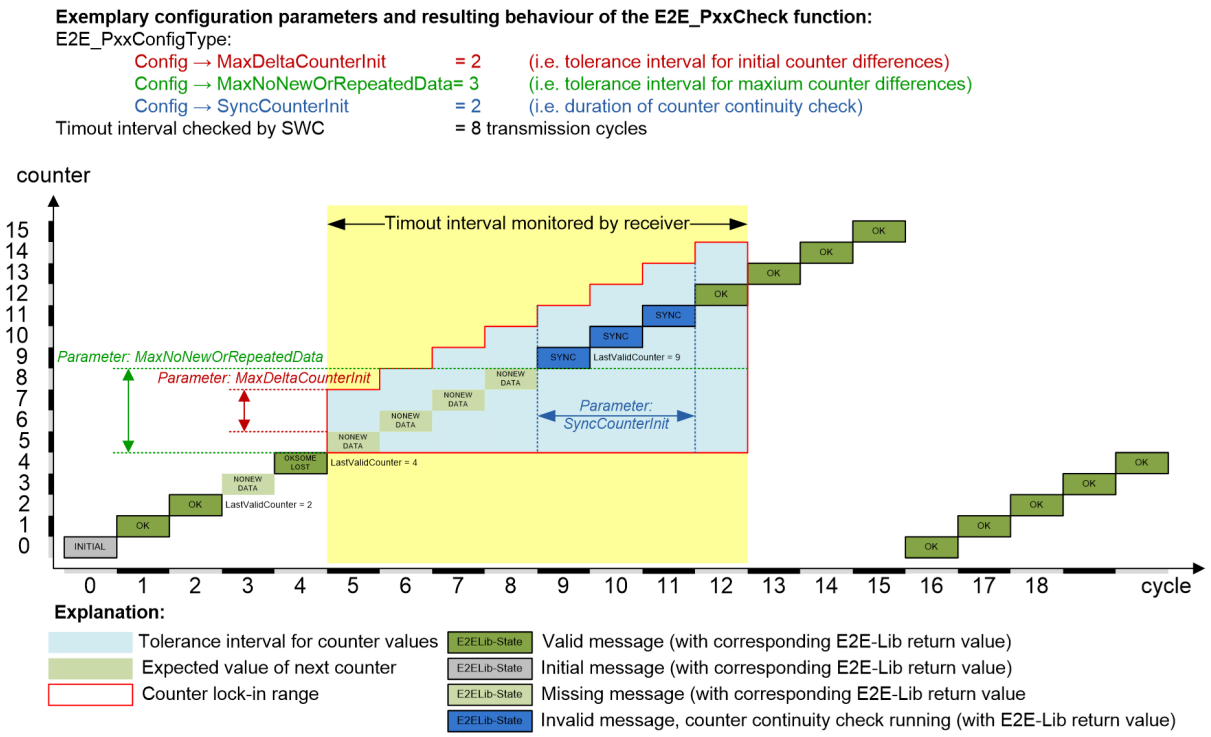


Figure 9.3: Configuration parameters of the E2E_PxxCheck() function and their effects

Clarification regarding SYNC states in Figure 9.3: In cycle 9, the counter value is not trustable anymore since the NoNewOrRepeatedData exceeds MaxNoNewOrRepeated Data. The resulting behavior is similar to as if an "unexpected behavior of the counter" is detected in cycle 9. Thus, the "counter continuity check" spans from cycle 10-11.

[UC_E2E_00207]

Upstream requirements: RS_E2E_08528

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

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

[UC_E2E_00209]

Upstream requirements: RS_E2E_08528

⌈In every receive cycle, the Receiver shall:

- Invoke once the reception function Rte_Read_<p>_<o>().

1. Set the attribute State->NewDataAvailable to TRUE if new data has been received without any errors:
    - In case of single channel or channel 1: State->NewDataAvailable = (retRte Read == RTE_E_OK) ? TRUE : FALSE;

- In case of channel 2: State->NewDataAvailable = TRUE; (note: the second channel has no access to Rte_Read return value).

2. Update Data, using received data element or I-PDU.
3. Call once the function E2E_PXXCheck().
4. Handle results (return value and State parameter) returned by E2E_PXXCheck().

⌋

Note: In case of single channel only, the NewDataAvailable flag may additionally incorporate the return value of the Rte_IsUpdated() API (if available) in the following way:

1. Invoke once the function Rte_IsUpdated_<p>_<o>().
2. Distinguish
   - If Rte_IsUpdated_<p>_<o>() returned FALSE : Set the attribute State->NewDataAvailable to FALSE and retRteRead to RTE_E_OK
   - If Rte_IsUpdated_<p>_<o>() returned TRUE :
     - Invoke once the reception function Rte_Read_<p>_<o>()
     - Set the attribute State->NewDataAvailable to TRUE if Rte_Read_<p>_<o>() returned RTE_E_OK, otherwise set it to FALSE
3. Steps 3.-5. as stated in [UC_E2E_00209].

This resembles the optional functionality of E2EPW_Read_<p>_<o>() as specified in AR 3.2.1 - 3.2.2 / AR 4.0.1 - AR 4.1.1. It was changed as the functionality of RTE_IsUpdated_<p>_<o>() strongly depends on the underlying Com stack to provide a reliable reception indication (callback). Otherwise, corrupted data might be masked.

Note: In case of single channel only AND if AliveTimeout is configured, the NewDataAvailable flag may additionally incorporate the return value of the Rte_IsUpdated() API (if available) in the following way:

1. Invoke Rte_IsUpdated_<p>_<o>
2. Invoke Rte_Read_<p>_<o>
3. Distinguish:
   - Set State->NewDataAvailable to TRUE if Rte_Read_<p>_<o> returned TRUE AND Rte_IsUpdated_<p>_<o> returned TRUE
   - Set State->NewDataAvailable to FALSE otherwise

This way the AliveTimeout "RTE_E_MAX_AGE_EXCEEDED" is returned via Rte_Read_<p>_<o> to the SWC.

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

[UC_E2E_00233]

Upstream requirements: RS_E2E_08528

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

9.2.1 Receiver at data element level

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

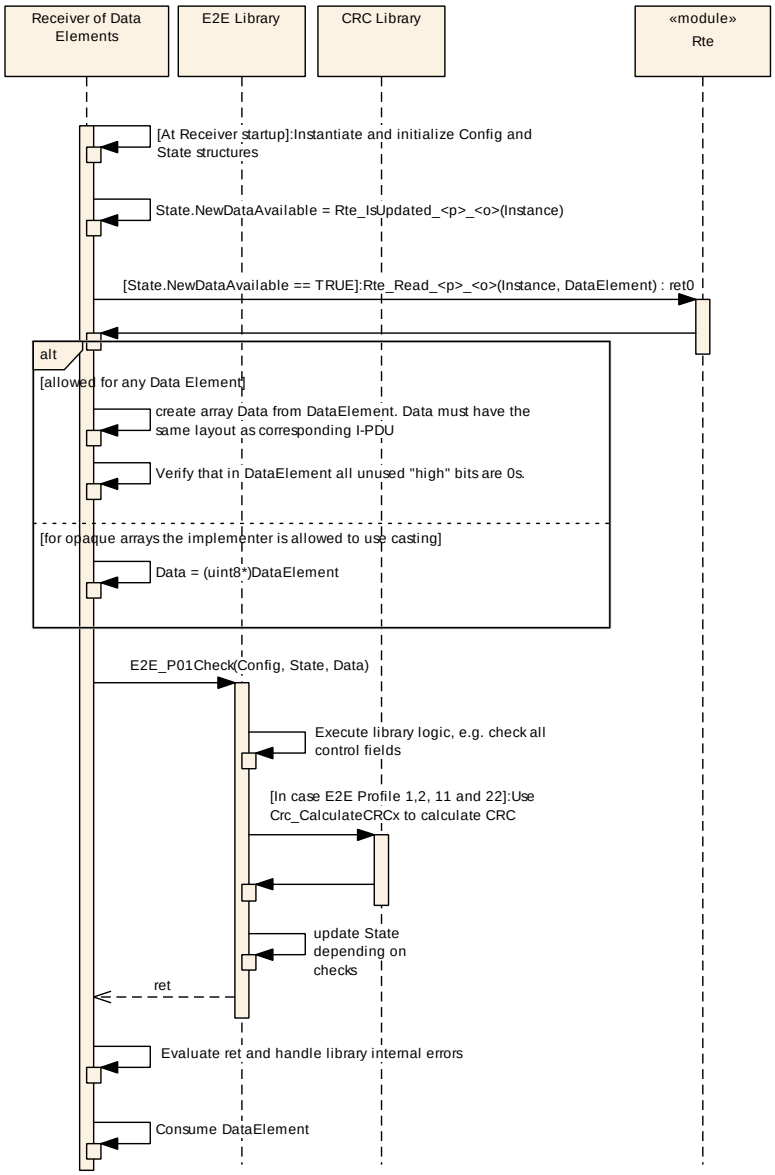


Figure 9.4: Receiver of data elements

[UC_E2E_00277]

Upstream requirements: RS_E2E_08528

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

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

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

[UC_E2E_00278]

Upstream requirements: RS_E2E_08528

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

9.2.2 Receiver at signal group level

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

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

Diagram below shows the step "State.".

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

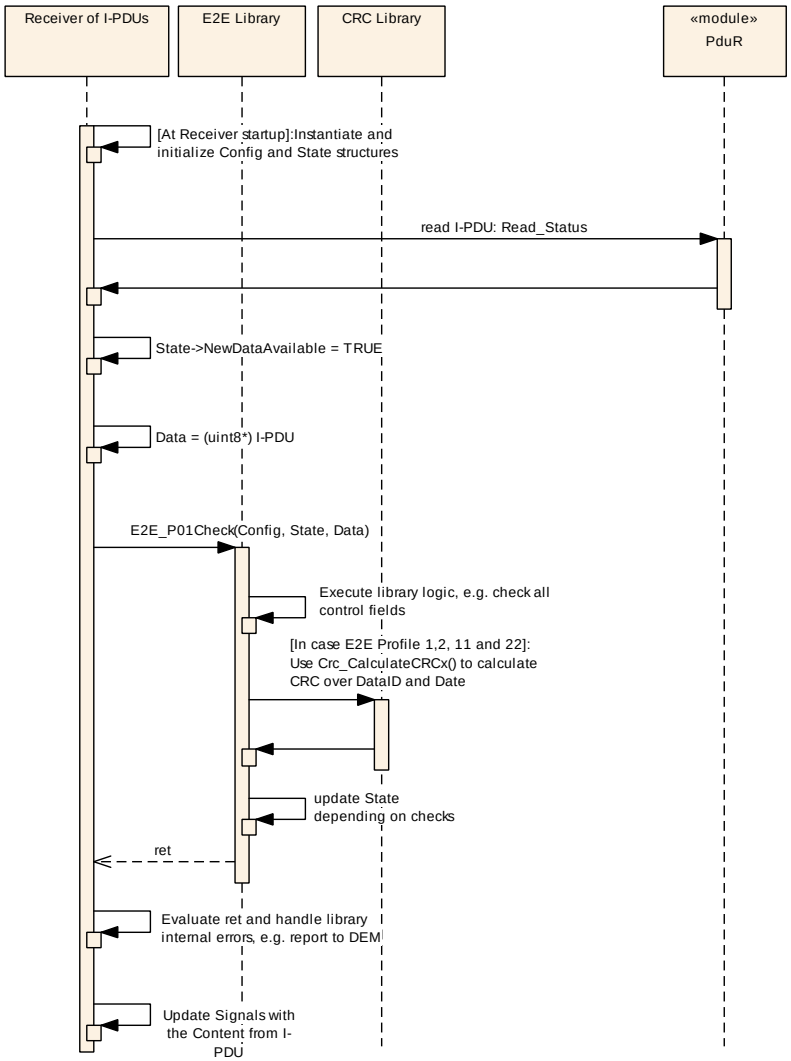


Figure 9.5: Receiver of I-PDUs

10 Configuration specification

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

[SWS_E2E_00037]

Upstream requirements: RS_E2E_08538

⌈The E2E library shall not have any configuration options.⌋

10.1 Published Information

[SWS_E2E_00038]

Upstream requirements: SRS_BSW_00402

⌈The standardized common published parameters as required by SRS_BSW_00402 in [5, General Requirements on Basic Software Modules] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in [2, General Specification of Basic Software Modules].⌋

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

A Annex A: Safety Manual for usage of E2E Library

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

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

A.1 E2E profiles and their standard variants

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

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

[UC_E2E_00053]

Upstream requirements: RS_E2E_08528

⌈Any user of E2E Profile 1 shall use whenever possible the defined E2E variants.⌋

A.2 E2E error handling

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

A general standardization of the error handing of an application is usually not possible.

[UC_E2E_00235]

Upstream requirements: RS_E2E_08528

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

A.3 Methodology of usage of E2E Library

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

In the first step the user selects the architectural approach how E2E Library is used in a given system (through COM callouts, through E2E Protection wrapper etc). There are several architectural solutions of usage of E2E Library described in Chapter Chapter A.5.

In the second step, the user selects which data elements or signal groups need to be protected and with which E2E Profile.In principle, all transmitted data identified as safety-related are those that need to be protected.

In the third step, the user determines the settings for each selected data element or signal group to be protected. The settings are stored in Software Component Template metaclass EndToEndDescription. The settings include e.g. Data ID, CRC offset.

1. For each signal group to be protected, there is a separate instance of EndToEnd Description, associated in System Template to ISignalIPdu metaclass.
2. For each data element to be protected, there is a separate instance of End ToEndDescription, associated indirectly to VariableDataPrototype, SenderCom Spec and ReceiverComSpec metaclasses.

In the fourth and last step, the user generates (or otherwise develops) the necessary glue code (e.g. E2E Protection Wrapper, COM callouts), responsible for invocation of E2E Library functions. The glue code serves as an adapter between the communication modules (e.g. COM, RTE) and E2E Library.

A.4 RTE configuration constraints for SW-C level protection

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

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

A.4.1 Communication model for SW-C level protection

AUTOSAR RTE supports different communication models, like client-server, sender-receiver, mode switch etc.

A.4.2 Multiplicities for SW-C level protection

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

[UC_E2E_00258]

Upstream requirements: RS_E2E_08528

⌈In case the E2E Library is used to protect data elements, then the selected multiplicity shall be 1:N or 1:1.⌋

A.4.3 Explicit access

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

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

- The receiver waits for new data: it is blocked/waiting until new data element from the sender arrives (RTE communication modes "wake up of wait point" and "activation of Runnable entity")
- The receiver gets the currently available data element from RTE, i.e. the most recent data element (RTE communication modes "Implicit data read access" and "Explicit data read access")

E2E Profile 1 and 2 together with the proposed E2E protection wrapper provide timeout detection (which is one of the failure modes to handle - e.g. message loss). This is achieved by having the receiver executing independently from the reception of the data, and by the usage of a counter within E2E Profiles. By this means, if e.g. a data element is lost, it is seen by the receiver that every time the read data element has the same counter. This however requires that the receiver is not solely executed uponthe arrival of data.

In case the receiver is event-driven, then a timeout mechanism at the receiver needs to be used. The timeout mechanism is not a part of E2E Library.

A.5 Restrictions on the use of COM features

The following table lists COM features with a brief description and provides a classification of restriction of use in combination with End-to-End communication protection as described in this document.

Note: This list only covers features of the BSW module COM in combination with E2E Library and E2E Protection Wrapper. It does not address features of above layers (e.g. RTE) or use-cases where the E2E Transformer is used.

The restriction classes are as follows:

- "supported" means that both (E2E COM Callout and E2EPW) do support this feature.
- "use case dependent" means that the feature might be used/usable depending on the actual use case and configuration on sender and receiver side. However, suitability for an actual system and its influence on the safety requirements has to be analysed.

- "not supported" means that at least one variant (either E2E COM Callout or E2EPW) does not support this feature or a failure mode can be masked.

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



△

COM Feature / brief description	Classification
[SRS_Com_02103] I-PDU Replication mechanism	use case dependent
[SRS_Com_02104] I-PDU replication configuration	use case dependent
[SRS_Com_02105] Transmission and reception using I-PDU Replication	use case dependent
[SRS_Com_02106] I-PDU Replication error handling	use case dependent
Minimum Delay Time	use case dependent
Filtering at receiver side (e.g. COM273)	use case dependent
Filtering at sender side	use case dependent
Multiple Signal groups within an I-PDU	use case dependent

Table A.1: Classification of COM features

B Annex B: Application hints on usage of E2E Library

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

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

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

The E2E Library can be used in different ways (each explained in a separate section of this chapter):

- COM callouts - non-standard integrator code to protect I-PDUs (Chapter B.1).
- Out-of-box protection at RTE level (E2E Transformer)(Chapter B.2)

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

This annex describes an exemplary,basic solution how E2E Library can be invoked. This can be done by means of dedicated COM Callouts invoking E2E Library to protect signal groups representing data elements (which is called COM E2E Callouts, see Chapter B.1).

All necessary options, enabling to generate the code for the described solutions are available in AUTOSAR configuration, defined in [6, System Template] and [7, Software Component Template]. This contains e.g. association of I-PDUs with Data IDs.

To generate the COM E2E callouts for an I-PDU, the user defines EndToEnd* metaclasses and associates them to ISignalIPdu metaclass (representing the I-PDU).

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

[UC_E2E_00271]

Upstream requirements: RS_E2E_08528

⌈A given I-PDU, if protected by E2E, shall be protected by only one E2E mechanism.⌋

B.1 COM E2E Callouts

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

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

The callout invokes the E2E Library,once for each E2E-protected signal group ina given I-PDU.

This solution can be used in the systems where the integrity of operation of COM and RTE is provided.

B.1.1 Functional overview

For each I-PDU, there is a separate callout function. Each I-PDU callout function "knows" if and how each signal group of the I-PDU needs to be protected/checked. This means that the callout invokes the E2E Library functions with appropriate settings and state parameters. The E2E Library does now "know" signal groups and their settings - entire information is passed as function parameters to E2E library functions.

On both receiver and sender side, if a callout returns TRUE, then COM continues. If a COM E2E Callout returns FALSE, then COM stops to process the given I-PDU (in this cycle). The COM E2E Callout returns FALSE if and only if there is an internal error, e.g. program flow error, data corruption error in E2E Lib.

The sender callout always TRUE if there are no runtime errors detected (e.g. wrong parameter), otherwise FALSE. The receiver callout receiver returns TRUE if there are no runtime errors detected and the result of the check is either E2E_P02STATUS_OK or E2E_P02STATUS_OKSOMELOST.

The diagram below summarizes the COM E2E Callout solution on the sender side. The SW-C is completely not impacted, and only additional activities in COM is invocation of the generated callout (step 6). If the return value from the callout is TRUE, then the IpduData modified by E2E Library is then transmitted by PDU router. If false, then COM stops further processing of this I-PDU in this cycle.

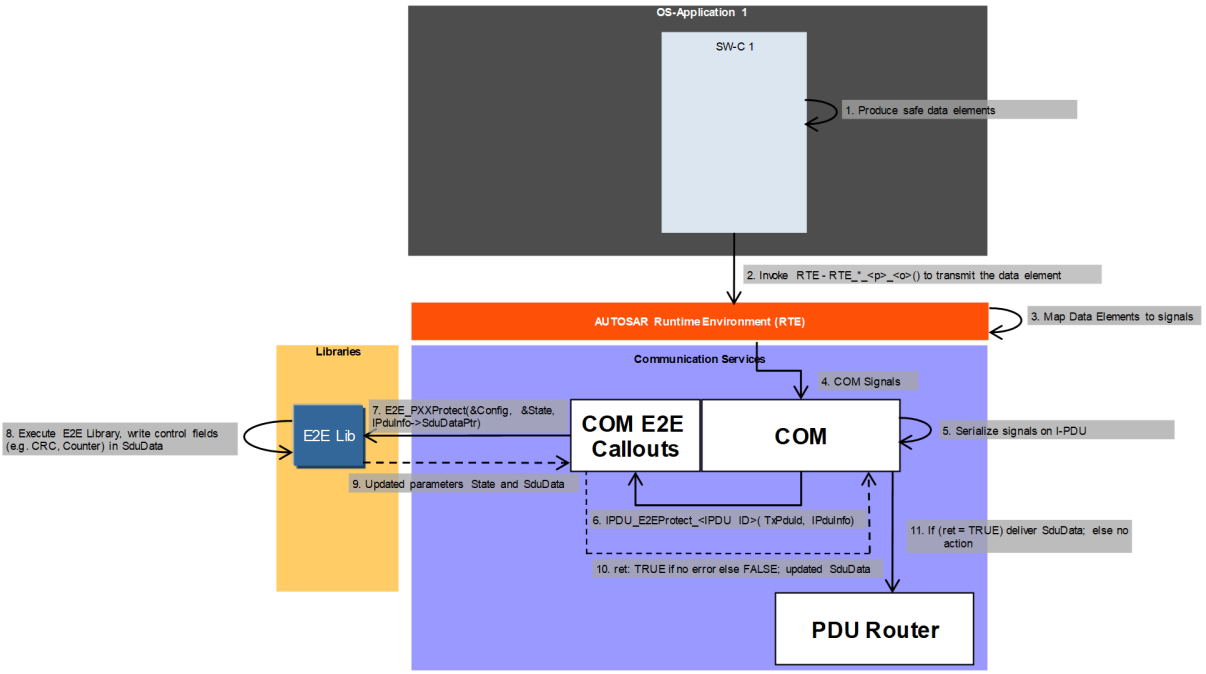


Figure B.1: Callout - overall flow - P-port

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

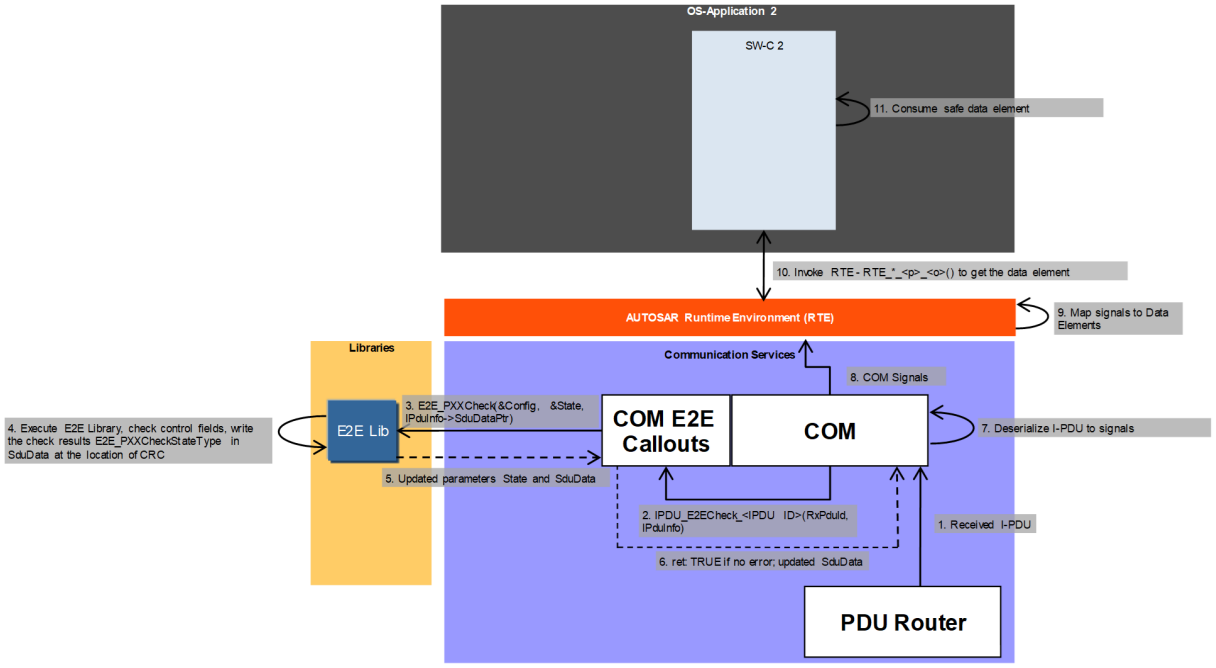


Figure B.2: Callout - overall flow - R-port

B.1.1.1 Sending/Calling

On the sender COM side, when the I-PDU has been built from signals and the conversions (e.g. Endianness) have taken place, and the I-PDU is ready, then COM calls a callout function. There is a separate callout for each I-PDU (if defined). Once the callout returns, COM invokes the PDU Router to transmit the data (fuction PduR_Com Transmit).

The callout function is generated to protect the signal groups of one I-PDU and simply invokes the E2E Library(once per each E2E-protected signal group) with the correct hard-coded settings. The hard-coded settings have been generated from the settings described in the previous section.

When the callout returns TRUE, COM invokes PduR_ComTransmit(), to route the I-PDU through the network.

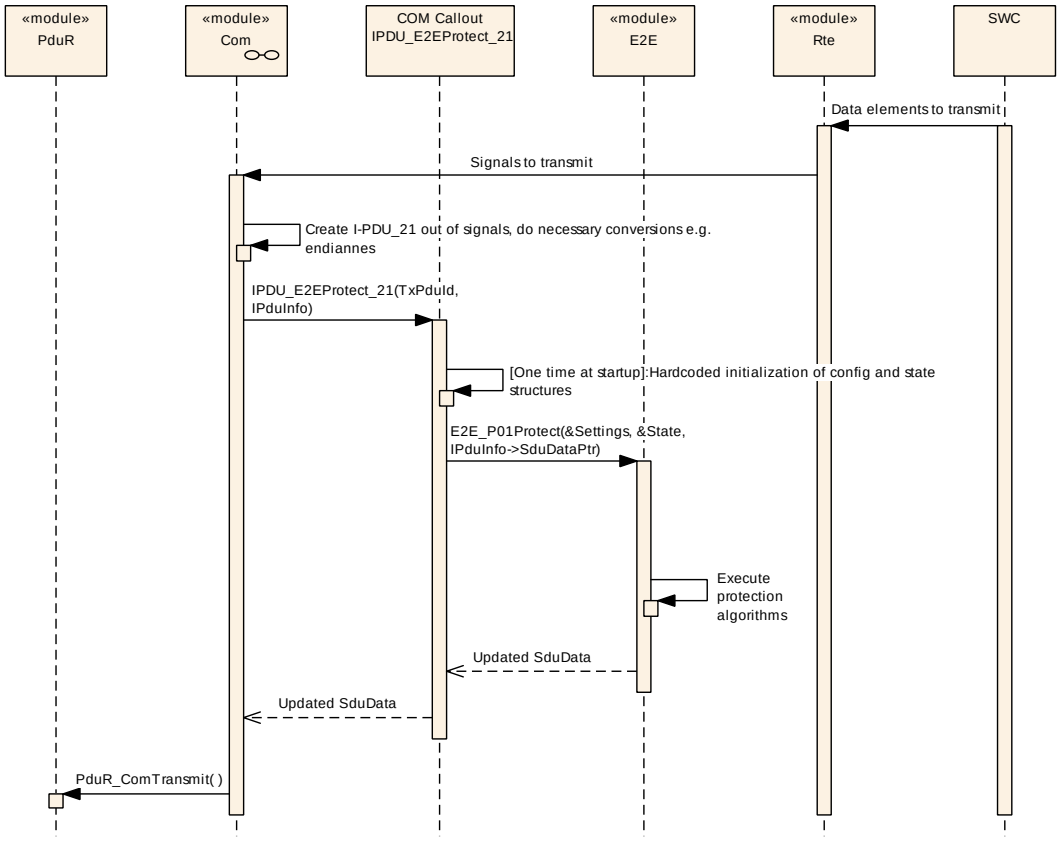


Figure B.3: Callout - sequence - sending

According to COM SWS, the callouts shall conform to the following syntax:

boolean <IPDU_CalloutName> (PduIdType TxPduId, PduInfoType* PduInfoPtr)

[UC_E2E_00250]

Upstream requirements: RS_E2E_08528

⌈The transmission callout for usage with E2E shall be the following: IPDU_ E2EProtect_<IPDU ID>(PduIdType TxPduId, PduInfoType* PduInfoPtr).

For example, the callout to protect the I-PDU with handle 21 shall have the name IPDU_E2EProtect_21().⌋

B.1.1.2 Reception

On the receiver COM side, when the I-PDU is available at PDU Router, PDU Router invokes COM's function COM_RxIndication(). COM then calls the generated I-PDU callout (if configured for the given I-PDU). The callout, generated specifically for that I-PDU, calls the E2E Library with specific parameters (once for each E2E-protected signal group). The E2E Library executes the checks and stores the check results in the status.

Once E2E Library check function returns, the callout copies the status into the CRC byte, so that it can be analyzed, if needed, by receiver SW-C.

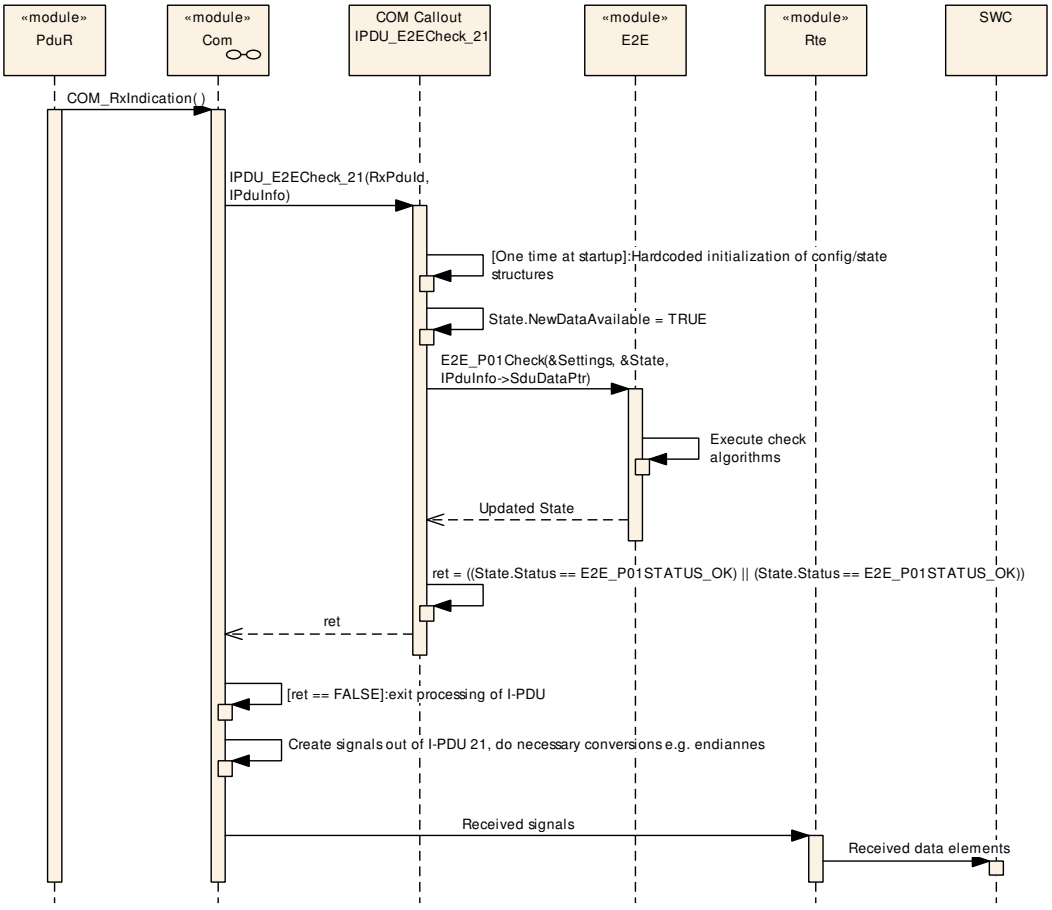


Figure B.4: Callout - sequence - reception

[UC_E2E_00251]

Upstream requirements: RS_E2E_08528

⌈The reception callout for usage with E2E shall be the following: IPDU_E2ECheck_<IPDU ID>(PduIdType RxPduId, const PduInfoType* PduInfoPtr).

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

B.1.2 Methodology

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

The information how each signal group needs to be protected (e.g. which E2E Profile, which offset) is defined in [6, System Template] , [7, Software Component Template] and [8, Specification of ECU Configuration]. This configuration information is used to generate the callout functions.

By means of the settings defined by AUTOSAR templates, it is possible to generate the COM callouts for invoking the E2E Library.

The configuration is done in the following configuration areas:

- Definition of I-PDUs (system template)
- Definition of E2E settings ([7, Software Component Template])
- Association of I-PDUs to E2E protection settings (system template).
- Definition of I-PDU details (ECU configuration)

The four above steps are described in more details below.

First, according to System Template, the I-PDUs exchanged by COM are defined.

Secondly, according to Software Component Template, for each signal group to be protected, the classes EndToEndProtection and EndToEndDescription are defined. The settings include information like CRC offset.

Thirdly, according to System Template, each I-PDU to be protected is associated to a corresponding EndToEndProtection.

Fourth, after the extraction of ECU configuration, according to ECU configuration, the I-PDU handles (numerical I-PDU identifers) and callout functions are defined. COM requires that there is a separate callout function for each I-PDU (separate piece of code).

Allconfiguration options needed to generate the COM callouts automatically is available in AUTOSAR methodology. For each I-PDU to be protected/checked, a separate callout routine shall be genrated, which invokes E2E Library (once or several times).

[UC_E2E_00270]

Upstream requirements: RS_E2E_08528

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

[UC_E2E_00290]

Upstream requirements: RS_E2E_08528

⌈If the E2EProtection is done via COM Callouts then the EndToEndProtectionISignal IPdu shall be defined.⌋

Note that in R3.2 (contrary to >=R4.0), the ISignalIPdu is called "SignalIPdu" and it inherits the unusedBitPattern attribute from IPdu.

The important settings are:

- ISignalIPdu (represents an I-PDU)
  - ISignalIPdu.unusedBitPattern: bits that are not used in an I-PDU,
- ISignalToIPduMapping: describes the mapping of signals to I-PDUs,
  - ISignalToIPduMapping.startPosition: offset in bits of a signal in the I-PDU,
- EndToEndProtectionISignalIPdu: association of one E2E protection to a one I-PDU and to one signal group,
  - EndToEndProtectionISignalIPdu.dataOffset: offset in bits of the signal group in the I-PDU.

ISignalIPdu.unusedBitPattern is not used by E2E COM callouts, because they are set by COM and E2E COM callouts operate on the same buffers.

B.1.3 Code Example

Note that the code examples for the COM E2E callouts are for the case when there is one signal group in the I-PDU. In general, it is possible to have N signal groups in an I-PDU and M signal groups protected by E2E, where 0<= M <= N. In such a case, the callout invokes E2E Library functions M times (for each of the protected signal group).

Transmitter

```
boolean IPDU_E2EProtect_21 (PduIdType RxPduId, PduInfoType* PduInfoPtr)
    {
     /* At first run, instantiate the structures and set the init Values
        */
     static E2E_P01ConfigType Cfg_Write_21 = { 64, 21,
        E2E_P01_DATAID_BOTH, 1, 0, 8 };
     static E2E_P01ProtectStateType Sta_Write_21 = {0};
```

```
    Std_ReturnType ret = E2E_P01Protect(& Cfg_Write_21, & Sta_Write_21,
        PduInfoPtr->SduDataPtr);
    /* return TRUE if no error in protect function */
    return (ret != 0);
}
```

Receiver

```
boolean IPDU_E2ECheck_21 (PduIdType RxPduId, const PduInfoType*
    PduInfoPtr) {
    /* At first run, instantiate the structures and set the init values
       */
    static E2E_P01ConfigType Cfg_Read_21 = { 64, 21,
       E2E_P01_DATAID_BOTH, 1, 0, 8 };
    static E2E_P01CheckStateType Sta_Read_21 = {0, 0, TRUE, FALSE,
       E2E_P01STATUS_NONEWDATA};
    /* If callout is invoked, this means that new data is available At
       COM */
    Sta_Read_21.NewDataAvailable = TRUE;
    Std_ReturnType ret = E2E_P01Check(Cfg_Read_21, Sta_Read_21,
       PduInfoPtr->SduDataPtr);

    /* return TRUE if no error, possibly only some messages lost Within
        counter tolerance */
    if(ret == E2E_OK && (Sta_Read_21.Status == E2E_P01STATUS_OK ||
       Sta_Read_21.Status == E2E_P02STATUS_OKSOMELOST) ) {
        return TRUE;
    }
    else {
        return FALSE;
    }
}
```

B.2 Protection at RTE level through E2E Transformer

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

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

This scenario is specified in details in SWS E2E Transformer.

C Not applicable requirements

[SWS_E2E_NA_00294]

Upstream requirements: SRS_BSW_00004, SRS_BSW_00101, SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00171, SRS_BSW_00336, SRS_BSW_00339, SRS_BSW_00344, SRS_BSW_00345, SRS_BSW_00369, SRS_BSW_00375, SRS_BSW_00380, SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00385, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00403, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00406, SRS_BSW_00409, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00419, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00450, SRS_BSW_00451, SRS_BSW_00452, SRS_BSW_00458, SRS_BSW_00461, SRS_BSW_00467, SRS_BSW_00469, SRS_BSW_00470, SRS_BSW_00471, SRS_BSW_00472, SRS_BSW_00478, SRS_BSW_00488, SRS_BSW_00489, SRS_BSW_00490, SRS_BSW_00491, SRS_BSW_00493, SRS_BSW_00496, SRS_BSW_00497

⌈These requirements are not applicable to this specification.⌋

D Change history of AUTOSAR traceable items

D.1 Traceable item history of this document according to AUTOSAR Release R22-11

D.1.1 Added Specification Items in R22-11

[SWS_E2E_00011] [SWS_E2E_00017] [SWS_E2E_00018] [SWS_E2E_00020]
[SWS_E2E_00021] [SWS_E2E_00022] [SWS_E2E_00032] [SWS_E2E_00033]
[SWS_E2E_00037] [SWS_E2E_00038] [SWS_E2E_00047] [SWS_E2E_00048]
[SWS_E2E_00049] [SWS_E2E_00050] [SWS_E2E_00110] [SWS_E2E_00115]
[SWS_E2E_00152] [SWS_E2E_00153] [SWS_E2E_00154] [SWS_E2E_00158]
[SWS_E2E_00160] [SWS_E2E_00161] [SWS_E2E_00166] [SWS_E2E_00200]
[SWS_E2E_00214] [SWS_E2E_00215] [SWS_E2E_00216] [SWS_E2E_00311]
[SWS_E2E_00314] [SWS_E2E_00318] [SWS_E2E_00319] [SWS_E2E_00320]
[SWS_E2E_00321] [SWS_E2E_00322] [SWS_E2E_00323] [SWS_E2E_00324]
[SWS_E2E_00325] [SWS_E2E_00334] [SWS_E2E_00335] [SWS_E2E_00336]
[SWS_E2E_00337] [SWS_E2E_00338] [SWS_E2E_00339] [SWS_E2E_00340]
[SWS_E2E_00342] [SWS_E2E_00343] [SWS_E2E_00344] [SWS_E2E_00347]
[SWS_E2E_00349] [SWS_E2E_00350] [SWS_E2E_00351] [SWS_E2E_00352]
[SWS_E2E_00353] [SWS_E2E_00370] [SWS_E2E_00371] [SWS_E2E_00373]
[SWS_E2E_00377] [SWS_E2E_00378] [SWS_E2E_00379] [SWS_E2E_00380]
[SWS_E2E_00381] [SWS_E2E_00382] [SWS_E2E_00383] [SWS_E2E_00384]
[SWS_E2E_00385] [SWS_E2E_00386] [SWS_E2E_00387] [SWS_E2E_00388]
[SWS_E2E_00389] [SWS_E2E_00390] [SWS_E2E_00391] [SWS_E2E_00392]
[SWS_E2E_00393] [SWS_E2E_00437] [SWS_E2E_00438] [SWS_E2E_00439]
[SWS_E2E_00440] [SWS_E2E_00441] [SWS_E2E_00443] [SWS_E2E_00444]
[SWS_E2E_00445] [SWS_E2E_00446] [SWS_E2E_00447] [SWS_E2E_00448]
[SWS_E2E_00449] [SWS_E2E_00450] [SWS_E2E_00451] [SWS_E2E_00452]
[SWS_E2E_00453] [SWS_E2E_00454] [SWS_E2E_00455] [SWS_E2E_00456]
[SWS_E2E_00457] [SWS_E2E_00458] [SWS_E2E_00459] [SWS_E2E_00460]
[SWS_E2E_00461] [SWS_E2E_00462] [SWS_E2E_00476] [SWS_E2E_00477]
[SWS_E2E_00542] [SWS_E2E_00544] [SWS_E2E_00545] [SWS_E2E_00546]
[SWS_E2E_00547] [SWS_E2E_00548] [SWS_E2E_00549] [SWS_E2E_00550]
[SWS_E2E_00551] [SWS_E2E_00552] [SWS_E2E_00553] [SWS_E2E_00554]
[SWS_E2E_00555] [SWS_E2E_00556] [SWS_E2E_00557] [SWS_E2E_00558]
[SWS_E2E_00559] [SWS_E2E_00560] [SWS_E2E_00561] [SWS_E2E_00562]
[SWS_E2E_00563] [SWS_E2E_00564] [SWS_E2E_00565] [SWS_E2E_00567]
[SWS_E2E_00568] [SWS_E2E_00569] [SWS_E2E_00570] [SWS_E2E_00571]
[SWS_E2E_00572] [SWS_E2E_00573] [SWS_E2E_00574] [SWS_E2E_00575]
[SWS_E2E_00576] [SWS_E2E_00577] [SWS_E2E_00578] [SWS_E2E_00579]
[SWS_E2E_00580] [SWS_E2E_00581] [SWS_E2E_00583] [SWS_E2E_00584]
[SWS_E2E_00585] [SWS_E2E_00586] [SWS_E2E_00587] [SWS_E2E_00588]
[SWS_E2E_00589] [SWS_E2E_00590] [SWS_E2E_00591] [SWS_E2E_00592]
[SWS_E2E_00593] [SWS_E2E_00594] [SWS_E2E_00595] [SWS_E2E_00596]
[SWS_E2E_00597] [SWS_E2E_00598] [SWS_E2E_00599] [SWS_E2E_10001]

[SWS_E2E_10002] [SWS_E2E_10003] [SWS_E2E_10004] [SWS_E2E_10005] [SWS_E2E_10006] [SWS_E2E_10007] [SWS_E2E_91001] [SWS_E2E_91002] [SWS_E2E_91003] [SWS_E2E_91004] [SWS_E2E_91005] [SWS_E2E_91006] [SWS_E2E_91007] [SWS_E2E_91008] [SWS_E2E_91009] [SWS_E2E_91010] [SWS_E2E_91011] [SWS_E2E_91012] [SWS_E2E_91013] [SWS_E2E_91014] [SWS_E2E_91015] [SWS_E2E_91016] [SWS_E2E_91017] [SWS_E2E_91018] [SWS_E2E_91019] [SWS_E2E_91020] [SWS_E2E_91021] [SWS_E2E_91022] [SWS_E2E_91023] [SWS_E2E_91024] [SWS_E2E_91025] [SWS_E2E_91026] [SWS_E2E_91027] [SWS_E2E_91028] [SWS_E2E_91029] [SWS_E2E_91030] [SWS_E2E_91031] [SWS_E2E_91032] [SWS_E2E_91033] [SWS_E2E_91034] [SWS_E2E_91035] [SWS_E2E_91036] [SWS_E2E_91037] [SWS_E2E_91038] [SWS_E2E_91039] [SWS_E2E_91040] [SWS_E2E_91041] [SWS_E2E_91042] [SWS_E2E_91053] [SWS_E2E_91059] [SWS_E2E_91060] [SWS_E2E_91063] [SWS_E2E_91068] [SWS_E2E_91070] [SWS_E2E_91071] [SWS_E2E_91072] [SWS_E2E_91073] [SWS_E2E_91074] [SWS_E2E_91075] [SWS_E2E_91076] [SWS_E2E_91077] [SWS_E2E_91078] [SWS_E2E_91079] [SWS_E2E_91080] [SWS_E2E_91081] [SWS_E2E_91082] [SWS_E2E_91083] [SWS_E2E_91084] [SWS_E2E_91085] [SWS_E2E_91086] [SWS_E2E_91087] [SWS_E2E_91088] [SWS_E2E_91089] [SWS_E2E_91090] [SWS_E2E_91091] [SWS_E2E_91092] [SWS_E2E_91093] [SWS_E2E_91094] [SWS_E2E_NA_00294] [UC_E2E_00053] [UC_E2E_00089] [UC_E2E_00165] [UC_E2E_00192] [UC_E2E_00202] [UC_E2E_00203] [UC_E2E_00204] [UC_E2E_00205] [UC_E2E_00206] [UC_E2E_00207] [UC_E2E_00209] [UC_E2E_00213] [UC_E2E_00230] [UC_E2E_00232] [UC_E2E_00233] [UC_E2E_00235] [UC_E2E_00239] [UC_E2E_00242] [UC_E2E_00248] [UC_E2E_00249] [UC_E2E_00250] [UC_E2E_00251] [UC_E2E_00256] [UC_E2E_00257] [UC_E2E_00258] [UC_E2E_00261] [UC_E2E_00262] [UC_E2E_00263] [UC_E2E_00264] [UC_E2E_00265] [UC_E2E_00266] [UC_E2E_00267] [UC_E2E_00268] [UC_E2E_00270] [UC_E2E_00271] [UC_E2E_00272] [UC_E2E_00273] [UC_E2E_00274] [UC_E2E_00275] [UC_E2E_00277] [UC_E2E_00278] [UC_E2E_00279] [UC_E2E_00280] [UC_E2E_00288] [UC_E2E_00289] [UC_E2E_00290] [UC_E2E_00292] [UC_E2E_00293] [UC_E2E_00296] [UC_E2E_00297] [UC_E2E_00300] [UC_E2E_00301] [UC_E2E_00302] [UC_E2E_00303] [UC_E2E_00304] [UC_E2E_00313] [UC_E2E_00328]

D.1.2 Changed Specification Items in R22-11

none

D.1.3 Deleted Specification Items in R22-11

none

D.2 Traceable item history of this document according to AUTOSAR Release R23-11

D.2.1 Added Specification Items in R23-11

[SWS_E2E_00566] [SWS_E2E_00600] [SWS_E2E_00601] [SWS_E2E_00602] [SWS_E2E_00603] [SWS_E2E_00604] [SWS_E2E_00605] [SWS_E2E_00606] [SWS_E2E_00607] [SWS_E2E_00608] [SWS_E2E_00609] [SWS_E2E_00610] [SWS_E2E_00611] [SWS_E2E_00612] [SWS_E2E_00613] [SWS_E2E_00614] [SWS_E2E_00615]

D.2.2 Changed Specification Items in R23-11

[SWS_E2E_00572] [SWS_E2E_00589] [SWS_E2E_91088] [SWS_E2E_91090] [SWS_E2E_91091] [SWS_E2E_91092]

D.2.3 Deleted Specification Items in R23-11

none

D.3 Traceable item history of this document according to AUTOSAR Release R24-11

D.3.1 Added Specification Items in R24-11

[SWS_E2E_00616] [SWS_E2E_00617] [SWS_E2E_00618] [SWS_E2E_00619] [SWS_E2E_00623] [SWS_E2E_00624] [SWS_E2E_00625] [SWS_E2E_00626] [SWS_E2E_00627] [SWS_E2E_91095] [SWS_E2E_91096] [SWS_E2E_91098] [SWS_E2E_91099] [SWS_E2E_91100] [SWS_E2E_91101] [SWS_E2E_91102]

D.3.2 Changed Specification Items in R24-11

[SWS_E2E_00038] [SWS_E2E_00314] [SWS_E2E_00318] [SWS_E2E_00319] [SWS_E2E_00320] [SWS_E2E_00321] [SWS_E2E_00322] [SWS_E2E_00323] [SWS_E2E_00324] [SWS_E2E_00325] [SWS_E2E_00343] [SWS_E2E_00571] [SWS_E2E_00584] [SWS_E2E_00589] [SWS_E2E_91029] [SWS_E2E_91038] [UC_E2E_00089] [UC_E2E_00165] [UC_E2E_00192] [UC_E2E_00213] [UC_E2E_00239] [UC_E2E_00242] [UC_E2E_00248] [UC_E2E_00249] [UC_E2E_00256] [UC_E2E_00257] [UC_E2E_00261] [UC_E2E_00262] [UC_E2E_00263] [UC_E2E_00264] [UC_E2E_00265] [UC_E2E_00266] [UC_E2E_00267] [UC_E2E_00268] [UC_E2E_00272] [UC_E2E_00273] [UC_E2E_00274] [UC_E2E_00275] [UC_E2E_00279]

[UC_E2E_00280] [UC_E2E_00288] [UC_E2E_00289] [UC_E2E_00292] [UC_E2E_00293] [UC_E2E_00296] [UC_E2E_00297] [UC_E2E_00300] [UC_E2E_00301] [UC_E2E_00302] [UC_E2E_00303] [UC_E2E_00304] [UC_E2E_00328]

D.3.3 Deleted Specification Items in R24-11

none

D.4 Traceable item history of this document according to AUTOSAR Release R25-11

D.4.1 Added Specification Items in R25-11

[SWS_E2E_91103] [SWS_E2E_91104] [SWS_E2E_91105] [SWS_E2E_91106] [SWS_E2E_91107] [SWS_E2E_91108] [SWS_E2E_91109] [SWS_E2E_91110] [SWS_E2E_91111] [SWS_E2E_91112] [SWS_E2E_91113] [SWS_E2E_91114] [SWS_E2E_91115] [SWS_E2E_91116] [SWS_E2E_91117] [SWS_E2E_91118] [SWS_E2E_91119] [SWS_E2E_91120] [SWS_E2E_91121] [SWS_E2E_91122]

D.4.2 Changed Specification Items in R25-11

[SWS_E2E_00037] [SWS_E2E_00038] [SWS_E2E_00047] [SWS_E2E_00049] [SWS_E2E_00158] [SWS_E2E_00160] [SWS_E2E_00161] [SWS_E2E_00166] [SWS_E2E_00216] [SWS_E2E_00334] [SWS_E2E_00338] [SWS_E2E_00339] [SWS_E2E_00340] [SWS_E2E_00393] [SWS_E2E_00437] [SWS_E2E_00441] [SWS_E2E_00446] [SWS_E2E_00449] [SWS_E2E_00457] [SWS_E2E_00544] [SWS_E2E_00546] [SWS_E2E_00547] [SWS_E2E_00548] [SWS_E2E_00549] [SWS_E2E_00565] [SWS_E2E_00583] [SWS_E2E_00584] [SWS_E2E_00585] [SWS_E2E_00586] [SWS_E2E_00587] [SWS_E2E_00588] [SWS_E2E_00589] [SWS_E2E_00590] [SWS_E2E_00616] [SWS_E2E_91002] [SWS_E2E_91003] [SWS_E2E_91005] [SWS_E2E_91007] [SWS_E2E_91010] [SWS_E2E_91014] [SWS_E2E_91016] [SWS_E2E_91017] [SWS_E2E_91018] [SWS_E2E_91021] [SWS_E2E_91022] [SWS_E2E_91023] [SWS_E2E_91027] [SWS_E2E_91029] [SWS_E2E_91030] [SWS_E2E_91033] [SWS_E2E_91036] [SWS_E2E_91037] [SWS_E2E_91038] [SWS_E2E_91039] [SWS_E2E_91073] [SWS_E2E_91077] [SWS_E2E_91081] [SWS_E2E_91083] [SWS_E2E_91084] [SWS_E2E_91085] [SWS_E2E_91088] [SWS_E2E_91090] [SWS_E2E_91091] [SWS_E2E_91092] [SWS_E2E_91095] [SWS_E2E_91099] [SWS_E2E_91100] [SWS_E2E_91102] [UC_E2E_00251]

D.4.3 Deleted Specification Items in R25-11

[SWS_E2E_00314] [SWS_E2E_00318] [SWS_E2E_00319] [SWS_E2E_00320] [SWS_E2E_00321] [SWS_E2E_00322] [SWS_E2E_00323] [SWS_E2E_00324] [SWS_E2E_00325] [SWS_E2E_91096] [UC_E2E_00089] [UC_E2E_00165] [UC_E2E_00192] [UC_E2E_00213] [UC_E2E_00239] [UC_E2E_00242] [UC_E2E_00248] [UC_E2E_00249] [UC_E2E_00256] [UC_E2E_00257] [UC_E2E_00261] [UC_E2E_00262] [UC_E2E_00263] [UC_E2E_00264] [UC_E2E_00265] [UC_E2E_00266] [UC_E2E_00267] [UC_E2E_00268] [UC_E2E_00272] [UC_E2E_00273] [UC_E2E_00274] [UC_E2E_00275] [UC_E2E_00279] [UC_E2E_00280] [UC_E2E_00288] [UC_E2E_00289] [UC_E2E_00292] [UC_E2E_00293] [UC_E2E_00296] [UC_E2E_00297] [UC_E2E_00300] [UC_E2E_00301] [UC_E2E_00302] [UC_E2E_00303] [UC_E2E_00304] [UC_E2E_00328]

D.5 Traceable item history of this document according to AUTOSAR Release R26-11

D.5.1 Added Specification Items in R26-11

none

D.5.2 Changed Specification Items in R26-11

[SWS_E2E_00017] [SWS_E2E_00037] [SWS_E2E_00038] [SWS_E2E_00216] [SWS_E2E_91068]

D.5.3 Deleted Specification Items in R26-11

[SWS_E2E_00032] [SWS_E2E_00033]