Document Title	E2E Protocol Specification
Document Category	Protocol specification (PRS)
Document UID	849
Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Write/Read CRC steps for Profile 5 updated to use profile-specific byte order
2025-11-27	R25-11	AUTOSAR Release Management	• Constraints and use cases removed or changed to spec items • Check of E2E counter at server side for methods removed
2024-11-27	R24-11	AUTOSAR Release Management	• E2EPW Support removed • Profile 76 added
2023-11-23	R23-11	AUTOSAR Release Management	• Description of E2E state machine reworked
2022-11-24	R22-11	AUTOSAR Release Management	• New chapter 6.3 added with generalized flowcharts and SWS items for non method profiles. • Protocol specific flowcharts and SWS items for method (P04m and P07m) and non method profiles replaced by generalized flowcharts and items • Use consistent function names (e.g. E2EXf_handling_PXXm_server, E2EXf_handling_PXXm_client) • Corrections of Min/MaxDataLength in P04m, P07m and P08m • Corrections of errors in state machine specification • Counter handling for client/server communication updated

△

2021-11-25	R21-11	AUTOSAR Release Management	• New profiles 08m,44m • New protocol independent flowcharts
2020-11-30	R20-11	AUTOSAR Release Management	• E2E for methods. • New profiles 08,44,4m,7m • Extension of E2E State Machine
2019-11-28	R19-11	AUTOSAR Release Management	• Introduction of Constraints for Client-Server Communication. • Added E2E_PXXForward functionality to provide a mechanism for replicating received E2E Errors. • Incorporated new configuration options for switching between valid and invalid state of E2E State Machine. • Fixed interoperability issues between P01 and P11, P02 and P22. • Changed Document Status from Final to published.
2019-03-29	1.5.1	AUTOSAR Release Management	• clarification on choosing suitable maximum data lengths for E2E profiles.
2018-10-31	1.5.0	AUTOSAR Release Management	• Migrated all functional specifications from Classic Platform's SWS E2ELibrary into Foundation's E2E Protocol Specification • Moved all figures and tables out of specifications and added references to them • Fixed duplicate/missing figures in profiles 2 (Calculate DeltaCounter), 5 (Read CRC), 6 (Read Counter) and 11 (Read DataIDNibble). • Added protocol examples for each profile
2018-03-29	1.4.0	AUTOSAR Release Management	• No content changes
2017-12-08	1.3.0	AUTOSAR Release Management	• No content changes

▽

△

2017-10-27	1.2.0	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 13
2 Acronyms and Abbreviations 16
3 Related documentation 17
3.1 Input documents . . . 17
3.2 Standards and Norms . . . 17
4 Constraints and assumptions 18
4.1 Limitations . . . 18
4.1.1 Limitations in general . . . 18
4.1.2 Limitations in signal based communication . . . 18
4.1.3 Limitations in service oriented communication with events . . . 19
4.1.4 Limitations in service oriented communication in Client/Server architecture . . . 19
4.1.5 Signal to Service Translation . . . 20
4.2 Applicability to car domains . . . 20
5 Requirements Tracing 21
6 Functional specification 27
6.1 Overview of communication protection . . . 27
6.2 Overview of E2E Profiles . . . 27
6.2.1 Error detection . . . 29
6.2.2 Common Types of E2E Profiles . . . 29
6.2.2.1 Profile Xm Message Type Enumeration . . . 29
6.2.2.2 Profile Xm Message Result Enumeration . . . 30
6.2.3 General Functionality of an E2E-Profile . . . 30
6.2.3.1 Functionality of the Counter . . . 31
6.2.3.2 Timeout detection . . . 31
6.2.3.3 Cyclic Redundancy Check . . . 32
6.3 Specification of E2E Profiles - Generalized Part . . . 32
6.3.1 Counter . . . 33
6.3.2 Data ID . . . 33
6.3.3 Length . . . 34
6.3.4 CRC . . . 34
6.3.5 Timeout detection . . . 34
6.3.6 Creation of E2E-Header . . . 34
6.3.6.1 E2E_PXXProtect() . . . 34
6.3.6.2 E2E_PXXForward() . . . 39
6.3.7 Evaluation of the E2E-Header . . . 42
6.3.7.1 E2E_PXXCheck() . . . 42
6.3.8 Get Header Info . . . 47
6.3.9 Profile Data Types . . . 48

6.3.9.1 Profile Protect State Type . . . 48
6.3.9.2 Profile Check Status Type . . . 49
6.3.9.3 Profile Check Status Enumeration . . . 49
6.3.9.4 GetHeaderInfo Data Types . . . 50
6.4 Specification of E2E Profiles for Methods - Generalized Part . . . 50
6.4.1 Counter . . . 51
6.4.2 Data ID . . . 52
6.4.3 Length . . . 52
6.4.4 CRC . . . 52
6.4.5 Message Type . . . 53
6.4.6 Message Result . . . 53
6.4.7 Source ID . . . 54
6.4.8 Timeout detection . . . 54
6.4.9 Creation of the E2E header . . . 55
6.4.9.1 E2E_PXXmProtect() . . . 55
6.4.9.2 E2E_PXXmForward() . . . 61
6.4.10 Evaluation of the E2E Header . . . 65
6.4.10.1 E2E_PXXmSourceCheck() . . . 65
6.4.10.2 E2E_PXXmSinkCheck() . . . 72
6.4.11 Get Header Info . . . 77
6.4.12 Profile Data Types . . . 78
6.4.12.1 Profile XXm Protect State Type . . . 78
6.4.12.2 Profile XXm Check State Type . . . 79
6.4.12.3 Profile XXm Check Status Enumeration . . . 79
6.4.12.4 GetHeaderInfo Data Types . . . 80
6.5 Specification of E2E Profile 1 . . . 81
6.5.1 Header Layout . . . 82
6.5.1.1 Counter . . . 82
6.5.1.2 Data ID . . . 83
6.5.1.3 CRC calculation . . . 85
6.5.2 Creation of E2E-Header . . . 87
6.5.2.1 E2E_P01Protect . . . 87
6.5.2.2 Calculate CRC . . . 88
6.5.2.3 E2E_P01Forward . . . 91
6.5.3 Evaluation of E2E- Header . . . 94
6.5.3.1 E2E_P01Check . . . 94
6.5.4 Profile Data Types . . . 99
6.5.4.1 Profile 1 Protect State Type . . . 99
6.5.4.2 Profile 1 Check Status Type . . . 100
6.5.4.3 Profile 1 Check Status Enumeration . . . 101
6.5.4.4 Profile 1 Configuration Type . . . 103
6.5.5 E2E Profile 1 Protocol Examples . . . 104
6.5.5.1 DataIDMode set to E2E_P01_DATAID_ALT . . . 104

6.5.5.2 DataIDMode set to E2E_P01_DATAID_LOW . . . 105
6.5.5.3 DataIDMode set to E2E_P01_DATAID_NIBBLE . . . 105
6.6 Specification of E2E Profile 2 . . . 106
6.6.1 Header Layout . . . 107
6.6.1.1 Counter . . . 107
6.6.1.2 DataID . . . 108
6.6.1.3 CRC . . . 108
6.6.2 Creation of E2E-Header . . . 109
6.6.2.1 E2E_P02Protect . . . 109
6.6.2.2 E2E_P02Forward . . . 111
6.6.3 Evaluation of the E2E-Check . . . 113
6.6.3.1 E2E_P02Check . . . 113
6.6.4 Profile Data Types . . . 124
6.6.4.1 Profile 2 Protect State Type . . . 124
6.6.4.2 Profile 2 Check Status Type . . . 124
6.6.4.3 Profile 2 Check Status Enumeration . . . 125
6.6.4.4 Profile 2 Configuration Type . . . 126
6.6.5 E2E Profile 2 Protocol Examples . . . 127
6.7 Specification of E2E Profile 4 . . . 128
6.7.1 Header Layout . . . 129
6.7.2 Profile 4 Configuration Type . . . 130
6.7.3 E2E Profile 4 Protocol Examples . . . 130
6.8 Specification of E2E Profile 5 . . . 131
6.8.1 Header layout . . . 133
6.8.1.1 Counter . . . 133
6.8.1.2 Data ID . . . 133
6.8.1.3 Length . . . 134
6.8.1.4 CRC . . . 134
6.8.2 Creation of the E2E-Header . . . 134
6.8.2.1 E2E_P05Protect . . . 134
6.8.2.2 E2E_P05Forward . . . 137
6.8.3 Evaluation of the E2E-Header . . . 139
6.8.3.1 E2E_P05Check . . . 139
6.8.4 Profile Data Types . . . 140
6.8.4.1 Profile 5 Protect State Type . . . 140
6.8.4.2 Profile 5 Check Status Type . . . 140
6.8.4.3 Profile 5 Check Status Enumeration . . . 141
6.8.4.4 Profile 5 Configuration Type . . . 142
6.8.5 E2E Profile 5 Protocol Examples . . . 142
6.9 Specification of E2E Profile 6 . . . 143
6.9.1 Header layout . . . 144
6.9.1.1 Counter . . . 145
6.9.1.2 Data ID . . . 145

6.9.1.3 Length . . . 146
6.9.1.4 CRC . . . 146
6.9.2 Creation of E2E-Header . . . 146
6.9.2.1 E2E_P06Protect . . . 146
6.9.2.2 E2E_P06Forward . . . 148
6.9.3 Evaluation of E2E-Header . . . 150
6.9.3.1 E2E_P06Check . . . 150
6.9.4 Profile Data Types . . . 152
6.9.4.1 Profile 6 Protect State Type . . . 152
6.9.4.2 Profile 6 Check Status Type . . . 152
6.9.4.3 Profile 6 Check Status Enumeration . . . 153
6.9.4.4 Profile 6 Configuration Type . . . 154
6.9.5 E2E Profile 6 Protocol Examples . . . 154
6.10 Specification of E2E Profile 7 . . . 155
6.10.1 Header layout . . . 156
6.10.2 Profile 7 Configuration Type . . . 157
6.10.3 E2E Profile 7 Protocol Examples . . . 157
6.11 Specification of E2E Profile 8 . . . 159
6.11.1 Header layout . . . 159
6.11.2 Profile 8 Configuration Type . . . 160
6.11.3 E2E Profile 8 Protocol Examples . . . 161
6.12 Specification of E2E Profile 11 . . . 162
6.12.1 Header Layout . . . 163
6.12.1.1 Counter . . . 164
6.12.1.2 Data ID . . . 165
6.12.1.3 Length . . . 166
6.12.1.4 CRC . . . 166
6.12.2 Creation of the E2E-Header . . . 166
6.12.2.1 E2E_P11Protect . . . 166
6.12.2.2 E2E_P11Forward . . . 170
6.12.3 E2E_P11Check . . . 174
6.12.4 Profile 11 Data Types . . . 179
6.12.4.1 Profile 11 Protect State Type . . . 179
6.12.4.2 Profile 11 Check Status Type . . . 179
6.12.4.3 Profile 11 Check Status Enumeration . . . 180
6.12.4.4 Profile 11 Configuration Type . . . 180
6.12.5 E2E Profile 11 Protocol Examples . . . 181
6.12.5.1 DataIDMode set to E2E_P11DATAID_NIBBLE . . . 182
6.12.5.2 DataIDMode set to E2E_P11DATAID_NIBBLE, Offset set to 64 . . . 182
6.13 Specification of E2E Profile 22 . . . 183
6.13.1 Header layout . . . 184
6.13.1.1 Counter . . . 185
6.13.1.2 Data ID . . . 185

6.13.1.3 Length . . . 186
6.13.1.4 CRC . . . 186
6.13.2 Creation of E2E-Header . . . 186
6.13.2.1 E2E_P22Protect . . . 186
6.13.2.2 E2E_P22Forward . . . 189
6.13.3 Evaluation of E2E-Header . . . 191
6.13.3.1 E2E_P22Check . . . 191
6.13.4 Profile 22 Data Types . . . 193
6.13.4.1 Profile 22 Configuration Type . . . 193
6.13.5 E2E Profile 22 Protocol Examples . . . 194
6.13.5.1 Offset set to 64 . . . 195
6.14 Specification of E2E Profile 44 . . . 195
6.14.1 Header Layout . . . 196
6.14.2 Profile 44 Configuration Type . . . 197
6.14.3 E2E Profile 44 Protocol Examples . . . 197
6.15 Specification of E2E Profile 76 . . . 198
6.15.1 Header Layout . . . 199
6.15.1.1 Counter . . . 200
6.15.1.2 Length . . . 200
6.15.1.3 CRC . . . 200
6.15.2 Creation of the E2E-Header . . . 200
6.15.2.1 E2E_P76Protect . . . 200
6.15.3 Evaluation of the E2E-Header . . . 203
6.15.3.1 E2E_P76Check . . . 203
6.15.4 Profile Data Types . . . 206
6.15.4.1 Profile 76 Protect State Type . . . 206
6.15.4.2 Profile 76 Check State Type . . . 206
6.15.4.3 Profile 76 Check Status Enumeration . . . 207
6.15.4.4 Profile 76 Configuration Type . . . 208
6.15.5 E2E Profile 76 Protocol Examples . . . 208
6.16 Specification of E2E Profile 4m . . . 209
6.16.1 Header Layout . . . 210
6.16.2 Profile 4m Configuration Type . . . 211
6.16.3 E2E Profile 4m Protocol Examples . . . 211
6.16.4 Request Example . . . 212
6.16.5 Response Example . . . 212
6.16.6 Error Response Example . . . 213
6.17 Specification of E2E Profile 7m . . . 213
6.17.1 Header Layout . . . 214
6.17.2 Profile 7m Configuration Type . . . 215
6.17.3 E2E Profile 7m Protocol Examples . . . 216
6.17.4 Request Example . . . 216
6.17.5 Response Example . . . 217

6.17.6 Error Response Example . . . 217
6.18 Specification of E2E Profile 8m . . . 218
6.18.1 Header Layout . . . 219
6.18.2 Profile 8m Configuration Type . . . 220
6.18.3 E2E Profile 8m Protocol Examples . . . 220
6.18.4 Request Example . . . 221
6.18.5 Response Example . . . 221
6.18.6 Error Response Example . . . 222
6.19 Specification of E2E Profile 44m . . . 222
6.19.1 Header Layout . . . 223
6.19.2 Profile 44m Configuration Type . . . 224
6.19.3 E2E Profile 44m Protocol Examples . . . 224
6.19.4 Request Example . . . 225
6.19.5 Response Example . . . 225
6.19.6 Error Response Example . . . 226
6.20 Specification of E2E state machine . . . 226
6.20.1 Overview of the state machine . . . 226
6.20.2 State machine specification . . . 228
6.20.2.1 Transition from E2E_SM_NODATA . . . 232
6.20.2.2 Transition from E2E_SM_INIT . . . 232
6.20.2.3 Transition from E2E_SM_VALID . . . 233
6.20.2.4 Transition from E2E_SM_INVALID . . . 233
6.20.3 State Machine Types . . . 237
6.20.3.1 E2E State Machine Configuration Type . . . 237
6.20.3.2 E2E State Machine State Type . . . 238
6.20.3.3 E2E State Machine Status Enumeration . . . 239
6.20.3.4 Profile specific Check Status to State Machine Check Status Mappings . . . 240
6.20.4 FTTI and E2E Parameters . . . 243
7 E2E API specification 244
7.1 API of middleware to applications . . . 244
7.2 API of E2E . . . 245
8 Configuration Parameters 248
8.1 General Constraints . . . 250
8.1.1 E2E State Machine Settings . . . 250
9 Protocol usage and guidelines 252
9.1 E2E and SOME/IP . . . 252
9.2 Client-Server Communication . . . 253
9.3 Periodic use of E2E check . . . 254
9.4 Error handling . . . 254
9.5 Maximal lengths of Data, communication buses . . . 254
9.6 Functional Safety Requirements . . . 255

9.7 Message Layout . . . 256
9.7.1 Alignment of signals to byte limits . . . 256
9.7.2 Unused bits . . . 257
9.7.3 Byte order (Endianness) . . . 257
9.8 Configuration constraints on Data IDs . . . 258
9.8.1 Data IDs . . . 258
9.8.2 Double Data ID configuration of E2E Profile 1 and 11 . . . 259
9.8.3 Alternating Data ID configuration of E2E Profile 1 and 11 . . . 259
9.8.4 Nibble configuration of E2E Profile 1 and 11 . . . 259

A Usage and generation of DataID Lists for E2E profile 2 and 22 261
A.1 Example A (persistent routing error) . . . 261
A.1.1 Assumptions . . . 261
A.1.2 Solution . . . 262
A.1.3 Example B (forbidden configuration) . . . 263
A.2 Conclusion . . . 264
A.3 DataID List example . . . 264

B Change history of AUTOSAR traceable items 273
B.1 Traceable item history of this document according to AUTOSAR Release R19-11 . . . 273
B.1.1 Added Constraints . . . 273
B.1.2 Changed Constraints . . . 273
B.1.3 Deleted Constraints . . . 273
B.1.4 Added Specification Items . . . 274
B.1.5 Changed Specification Items . . . 274
B.1.6 Deleted Specification Items . . . 274
B.2 Traceable item history of this document according to AUTOSAR Release R20-11 . . . 274
B.2.1 Added Constraints in R20-11 . . . 274
B.2.2 Changed Constraints in R20-11 . . . 274
B.2.3 Deleted Constraints in R20-11 . . . 274
B.2.4 Added Specification Items in R20-11 . . . 274
B.2.5 Changed Specification Items in R20-11 . . . 281
B.2.6 Deleted Specification Items in R20-11 . . . 283
B.3 Traceable item history of this document according to AUTOSAR Release R21-11 . . . 284
B.3.1 Added Constraints in R21-11 . . . 284
B.3.2 Changed Constraints in R21-11 . . . 284
B.3.3 Deleted Constraints in R21-11 . . . 284
B.3.4 Added Specification Items in R21-11 . . . 285
B.3.5 Changed Specification Items in R21-11 . . . 287
B.3.6 Deleted Specification Items in R21-11 . . . 292
B.4 Traceable item history of this document according to AUTOSAR Release R22-11 . . . 292

B.4.1 Added Constraints in R22-11 . . . 292
B.4.2 Changed Constraints in R22-11 . . . 292
B.4.3 Deleted Constraints in R22-11 . . . 293
B.4.4 Added Specification Items in R22-11 . . . 293
B.4.5 Changed Specification Items in R22-11 . . . 294
B.4.6 Deleted Specification Items in R22-11 . . . 295
B.5 Traceable item history of this document according to AUTOSAR Release R23-11 . . . 301
B.5.1 Added Constraints in R23-11 . . . 301
B.5.2 Changed Constraints in R23-11 . . . 301
B.5.3 Deleted Constraints in R23-11 . . . 302
B.5.4 Added Specification Items in R23-11 . . . 302
B.5.5 Changed Specification Items in R23-11 . . . 302
B.5.6 Deleted Specification Items in R23-11 . . . 303
B.6 Traceable item history of this document according to AUTOSAR Release R24-11 . . . 303
B.6.1 Added Constraints in R24-11 . . . 303
B.6.2 Changed Constraints in R24-11 . . . 303
B.6.3 Deleted Constraints in R24-11 . . . 303
B.6.4 Added Specification Items in R24-11 . . . 303
B.6.5 Changed Specification Items in R24-11 . . . 306
B.6.6 Deleted Specification Items in R24-11 . . . 308
B.7 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 309
B.7.1 Added Constraints in R25-11 . . . 309
B.7.2 Changed Constraints in R25-11 . . . 309
B.7.3 Deleted Constraints in R25-11 . . . 309
B.7.4 Added Specification Items in R25-11 . . . 310
B.7.5 Changed Specification Items in R25-11 . . . 311
B.7.6 Deleted Specification Items in R25-11 . . . 312
B.8 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 313
B.8.1 Added Constraints in R26-11 . . . 313
B.8.2 Changed Constraints in R26-11 . . . 313
B.8.3 Deleted Constraints in R26-11 . . . 314
B.8.4 Added Specification Items in R26-11 . . . 314
B.8.5 Changed Specification Items in R26-11 . . . 314
B.8.6 Deleted Specification Items in R26-11 . . . 314

1 Introduction and functional overview

The concept of E2E communication protection assumes that safety-related [1] data exchange shall be protected at runtime against the effects of faults on the communication link (see Figure 1.1). Faults detected between a sender and a receiver using E2E communication protection include systematic software faults, such as faults that are introduced on the lower communication layers of sender or receiver, and random hardware faults introduced by the MCU hardware, communication peripherals, transceivers, communication lines or other communication infrastructure.

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 of the lower communication layers (e.g. RTE, IOC, COM and network stacks).

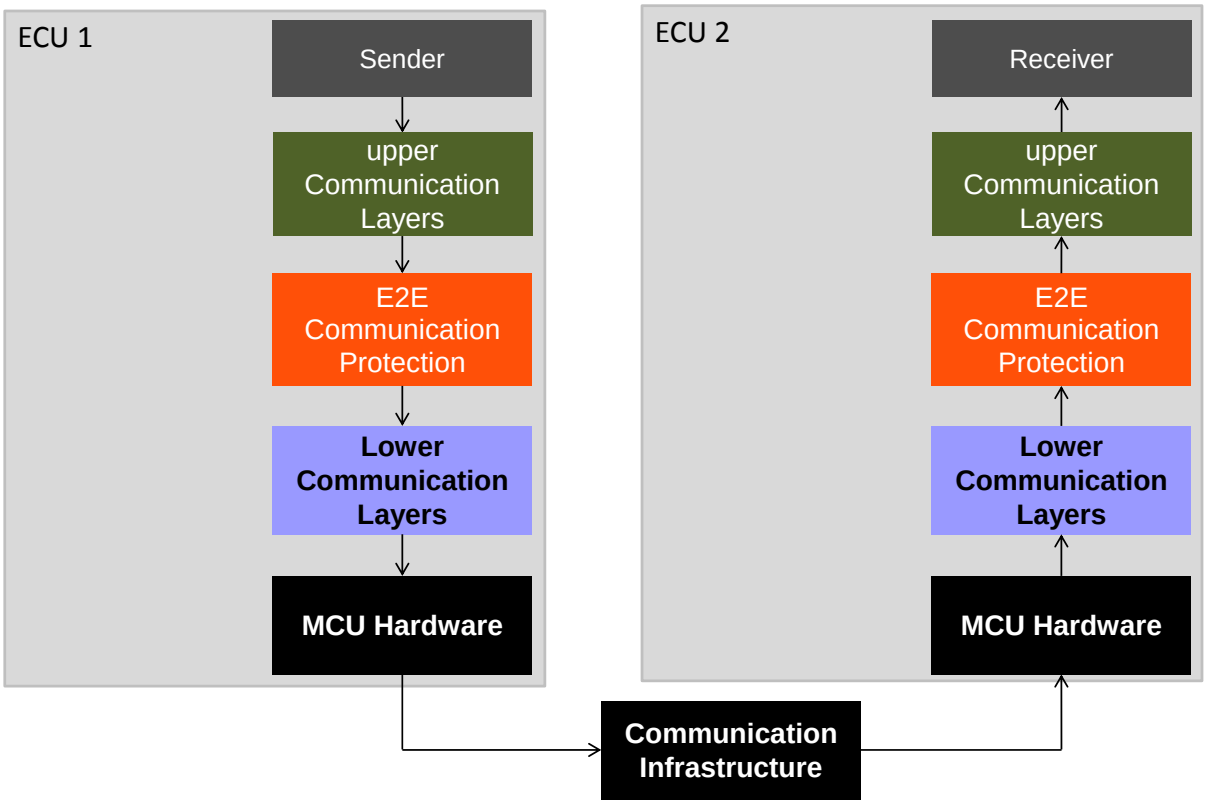

Figure 1.1: Overview of E2E communication protection between a sender and a receiver

By using E2E communication protection mechanisms, faults in lower software and hardware layers can be detected and handled at runtime. The E2E Supervision provides mechanisms for E2E communication protection, adequate for safety-related communication having requirements up to ASIL D.

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

The E2E communication protection allows the following:

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

To provide the appropriate solution addressing flexibility and standardization, AUTOSAR specifies a set of flexible E2E profiles that implement an appropriate combination of E2E communication protection mechanisms. Each specified E2E profile has a fixed set of mechanisms, as well as configuration options to configure the protocol header layout and status evaluation on the receiver side.

The E2E Supervision can be invoked from communication middleware e.g. from Adaptive Platform's ara::com, Classic Platform's RTE. It can be also invoked in a non-standardized way from other software, e.g. non-volatile memory managers, local IPCs, or intra-ECU bus stacks.

Appropriate usage of the E2E Supervision to fulfill the specific safety requirements for communication depends on several aspects. The specified profiles are capable, to a high probability, of detecting a large variety of communication faults. However, the use of a specific E2E profile requires the user to demonstrate that the selected profile provides sufficient error detection capabilities for the considered use case (taking into account various contributing factors, such as hardware failure rates, bit error rates, number of nodes in the network, repetition rate of messages, the usage of a gateway, potential software faults on the communication channel), as well as appropriate reaction on detected faults (e.g. by revoking repeated messages, determining timed-out communication or reacting on corrupt messages by initiating a safety reaction).

This specification specifies also the functionality, API and the configuration of the CRC routines.

The following routines for CRC calculation are specified:

- CRC8: SAEJ1850
- CRC8H2F: CRC8 0x2F polynomial
- CRC16
- CRC32
- CRC32P4: CRC32 0xF4ACFB13 polynomial
- CRC64: CRC-64-ECMA

For all routines (CRC8, CRC8H2F, CRC16, CRC32, CRC32P4 and CRC64), the following calculation methods are possible:

- Table based calculation: Fast execution, but larger code size (ROM table)
- Runtime calculation: Slower execution, but small code size (no ROM table)

- Hardware supported CRC calculation (device specific): Fast execution, less CPU time

All routines are re-entrant and can be used by multiple applications at the same time. Hardware supported CRC calculation may be supported by some devices in the future.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Communication Management that are not included in the [2, AUTOSAR glossary].

, **Abbreviation / Acronym:**	**Description:**
Data ID	An identifier that uniquely identifies the message / data element / data.
Source ID	An identifier that uniquely identifies the source of the message / data element / data (i.e., in case different sources produce the same message / data element / data – like different clients invoking the same method at a server).
Repetition	The same message was received more than once.
Loss	A message was not received.
Delay	A message was received later than expected.
Insertion	Unexpected information or an extra message was inserted.
Masquerade	non-authentic information is accepted as authentic information by a receiver.
Incorrect addressing	information is accepted from an incorrect sender or by an incorrect receiver.
Corruption	A communication fault, which changes information.
Asymmetric information	Receivers do receive different information from the same sender.
Subset	Information from a sender received by only a subset of the receivers.
Blocking	Blocking access to a communication channel.
FTTI	Fault tolerant time interval, maximum time a fault can be active before a hazard occurs.
tFD	Fault detection time interval. Time between occurence of a fault and its detection.
tFR	Fault reaction time interval. Time between fault detection and the reaction to the fault (switching to safe state or starting an emergency operation).

Table 2.1: Acronyms and Abbreviations

3 Related documentation

3.1 Input documents

[1] ISO 26262:2018 Road vehicles -– Functional Safety
https://www.iso.org

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] Specification of CRC Library
AUTOSAR_CP_SWS_CRCLibrary

[4] Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_CP_SWS_E2ELibrary

3.2 Standards and Norms

1. SAE-J1850 8-bit CRC
2. CCITT-FALSE 16-bit CRC. Refer to:

   ITU-T Recommendation X.25 (10/96) (Previously „CCITT Recommendation”)
   SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATION
   Public data networks - Interfaces
   Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit

   Section 2.2.7.4 „Frame Check Sequence (FCS) field” and Appendix I „Examples of data link layer transmitted bit patterns by the DCE and the DTE”
   `http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.25-199610-I!!PDF-E&type=items`
3. IEEE 802.3 Ethernet 32-bit CRC
4. ”32-Bit Cyclic Redundancy Codes for Internet Applications” [Koopman 2002]
5. Collection and evaluation of CRC polynomials by Philip Koopman, Carnegie Mellon University `https://users.ece.cmu.edu/~koopman/crc/`

4 Constraints and assumptions

4.1 Limitations

E2E communication protection is limited depending on the used type of communication. From E2E perspective the following types are distinguished:

- Signal based communication
- Service oriented communication with events
- Service oriented communication in Client/Server architecture
- Signal to Service Translation

In general, the behavior of the E2E protection mechanisms should be the same. However, some limitations exist depending on the communication type.

4.1.1 Limitations in general

- Communication errors which are detected at **lower layer** (e.g. by Ethernet FCS, IP header checksum, UDP checksum, SOME/IP header irregularities) will lead to **discarding the related message**. Thus, these messages might not reach the application at all.

4.1.2 Limitations in signal based communication

It is also called sender/receiver communication.

- E2E communication protection is limited to periodic or semi-periodic data communication paradigm, where the receiver has any expectancy on the regular reception of data and in case of communication loss/timeout or error, it performs an error handling.
- If one of E2E functions to protect (sender) or to check (receiver) is not called periodically, then some communication failure modes (loss, delay) may not be detected.
- If E2E protection is invoked at the level of data elements (e.g. from SW-Cs) and 1:N communication model is used and the data elements are sent using more than one IPDU, then all these I-PDUs shall have the same layout.
- Currently AUTOSAR does not provide the functionality to describe and handle more than one layout for the same data element (e.g. within the RTE) by using different protection mechanisms depending on Intra-ECU and Inter-ECU communication. Thus, for a 1:N sender-receiver relationship the user of E2E-protection is responsible to select one appropriate layout for the to be protected data elements. E.g. for a 1:N sender-receiver relationship the message layout can be

used for the transmission of data elements protected by protection to receivers located within and without the ECU.

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

4.1.3 Limitations in service oriented communication with events

It is also called event communication. (Note that here the name event is a bit confusing as a periodic communication is required. This is taken from the SOME/IP event.)

- E2E communication protection is limited to periodic or semi-periodic data communication paradigm, where the receiver has any expectancy on the regular reception of data and in case of communication loss/timeout or error, it performs an error handling.
- If one of the E2E functions to protect (sender) or to check (receiver) data is not called periodically, then some communication failure modes (loss, delay) may not be detected.

4.1.4 Limitations in service oriented communication in Client/Server architecture

The specified E2E communication protection for methods is limited to

- Communication between N clients and one dedicated server (N:1); this means, an E2E protected service is provided by exactly one server per system. In other words, if E2E protection is invoked at the level of data elements, then N:1 multiplicity, implicit communication, and remaining communication models (in particular client-server model) are not supported.

The specified E2E communication protection for methods may not detect all communication failure modes:

- If method requests are not invoked periodically, then some communication failure modes (loss, delay) may not be detected for received requests at the server.
- For C/S communication the following use cases need to be considered:

In case of using any PXX profile the defined E2E protection mechanisms assume no a priori knowledge at the server about the client that is requesting data (N:1 communication), communication failure modes (repetition, insertion) which are client specific may not be detected for received requests at the server. In this case, additional measures need to be implemented at application level to address those non-detected failure modes and complete E2E protection or arguments are to be provided showing that these failure modes are not relevant for a particular project.

In case of using E2E PXXm profiles, those failure modes could be covered. These profiles include the sourceID of the calling client which give the servers the possibility to distinguish them.

The values of the following E2E parameters are defined by the standard and shall not be changed.

- dataIdMode
- counterOffset
- crcOffset
- dataIdNibbleOffset
- offset

E2E Profiles 1, 2, 11 and 22 are not suggested to be used in Client-Server Communication.

4.1.5 Signal to Service Translation

Signal to service translation is limited to translation between sender/receiver and service oriented communication with events.

4.2 Applicability to car domains

The E2E supervision is applicable for the realization of safety-related automotive systems implemented by various SW-Cs distributed across different ECUs in a vehicle, interacting via communication links. The Supervision may also be used for intra-ECU communication (e.g. between memory partitions, processes, OSes/VMs in the same microcontroller, between CPU cores or microcontrollers).

5 Requirements Tracing

Requirement	Description	Satisfied by
[RS_E2E_08527]	Implementation of E2E protocol shall fulfill ISO 26262	[PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_01318] [PRS_E2E_01456]
[RS_E2E_08528]	E2E protocol shall provide different E2E profiles	[PRS_E2E_00012] [PRS_E2E_00075] [PRS_E2E_00076] [PRS_E2E_00085] [PRS_E2E_00117] [PRS_E2E_00118] [PRS_E2E_00119] [PRS_E2E_00120] [PRS_E2E_00121] [PRS_E2E_00122] [PRS_E2E_00123] [PRS_E2E_00124] [PRS_E2E_00125] [PRS_E2E_00126] [PRS_E2E_00127] [PRS_E2E_00128] [PRS_E2E_00129] [PRS_E2E_00130] [PRS_E2E_00132] [PRS_E2E_00133] [PRS_E2E_00134] [PRS_E2E_00135] [PRS_E2E_00136] [PRS_E2E_00137] [PRS_E2E_00138] [PRS_E2E_00139] [PRS_E2E_00140] [PRS_E2E_00141] [PRS_E2E_00142] [PRS_E2E_00143] [PRS_E2E_00145] [PRS_E2E_00146] [PRS_E2E_00147] [PRS_E2E_00148] [PRS_E2E_00149] [PRS_E2E_00150] [PRS_E2E_00151] [PRS_E2E_00163] [PRS_E2E_00169] [PRS_E2E_00190] [PRS_E2E_00195] [PRS_E2E_00196] [PRS_E2E_00298] [PRS_E2E_00299] [PRS_E2E_00300] [PRS_E2E_00301] [PRS_E2E_00306] [PRS_E2E_00400] [PRS_E2E_00420] [PRS_E2E_00508] [PRS_E2E_00526] [PRS_E2E_00540] [PRS_E2E_00541] [PRS_E2E_00588] [PRS_E2E_00589] [PRS_E2E_00591] [PRS_E2E_00592] [PRS_E2E_00594] [PRS_E2E_00596] [PRS_E2E_00597] [PRS_E2E_00598] [PRS_E2E_00599] [PRS_E2E_00600] [PRS_E2E_00608] [PRS_E2E_00609] [PRS_E2E_00610] [PRS_E2E_00611] [PRS_E2E_00612] [PRS_E2E_00613] [PRS_E2E_00614] [PRS_E2E_00641] [PRS_E2E_00644] [PRS_E2E_00645] [PRS_E2E_00646] [PRS_E2E_00647] [PRS_E2E_00648] [PRS_E2E_00651] [PRS_E2E_00652] [PRS_E2E_00653] [PRS_E2E_00654] [PRS_E2E_00655] [PRS_E2E_00656] [PRS_E2E_00657] [PRS_E2E_00660] [PRS_E2E_00661] [PRS_E2E_00662] [PRS_E2E_00663] [PRS_E2E_00666] [PRS_E2E_00667] [PRS_E2E_00668] [PRS_E2E_00669] [PRS_E2E_00670] [PRS_E2E_00673] [PRS_E2E_00677] [PRS_E2E_00706] [PRS_E2E_00735] [PRS_E2E_00739] [PRS_E2E_00743] [PRS_E2E_00851] [PRS_E2E_00852] [PRS_E2E_00855] [PRS_E2E_00856] [PRS_E2E_00857] [PRS_E2E_00858]

△

Requirement	Description	Satisfied by
		△ [PRS_E2E_00859] [PRS_E2E_00860] [PRS_E2E_00862] [PRS_E2E_00863] [PRS_E2E_00864] [PRS_E2E_00866] [PRS_E2E_00867] [PRS_E2E_00868] [PRS_E2E_00869] [PRS_E2E_00871] [PRS_E2E_00872] [PRS_E2E_00873] [PRS_E2E_00874] [PRS_E2E_00876] [PRS_E2E_00878] [PRS_E2E_00879] [PRS_E2E_00880] [PRS_E2E_00881] [PRS_E2E_00883] [PRS_E2E_00884] [PRS_E2E_00885] [PRS_E2E_00886] [PRS_E2E_00887] [PRS_E2E_00889] [PRS_E2E_00891] [PRS_E2E_00892] [PRS_E2E_00893] [PRS_E2E_00898] [PRS_E2E_00900] [PRS_E2E_00902] [PRS_E2E_00905] [PRS_E2E_00907] [PRS_E2E_00909] [PRS_E2E_00911] [PRS_E2E_00912] [PRS_E2E_00913] [PRS_E2E_00914] [PRS_E2E_00915] [PRS_E2E_00916] [PRS_E2E_00917] [PRS_E2E_00918] [PRS_E2E_00919] [PRS_E2E_00920] [PRS_E2E_00921] [PRS_E2E_01154] [PRS_E2E_01159] [PRS_E2E_01160] [PRS_E2E_01161] [PRS_E2E_01162] [PRS_E2E_01163] [PRS_E2E_01164] [PRS_E2E_01165] [PRS_E2E_01166] [PRS_E2E_01199] [PRS_E2E_01200] [PRS_E2E_01201] [PRS_E2E_01202] [PRS_E2E_01203] [PRS_E2E_01206] [PRS_E2E_01250] [PRS_E2E_01251] [PRS_E2E_01252] [PRS_E2E_01401] [PRS_E2E_01409] [PRS_E2E_01410] [PRS_E2E_01411] [PRS_E2E_01412] [PRS_E2E_01413] [PRS_E2E_01414] [PRS_E2E_01415] [PRS_E2E_01416] [PRS_E2E_01447] [PRS_E2E_01448] [PRS_E2E_01449] [PRS_E2E_01450] [PRS_E2E_01451] [PRS_E2E_01452] [PRS_E2E_01453] [PRS_E2E_01454] [PRS_E2E_01455] [PRS_E2E_01457] [PRS_E2E_01458] [PRS_E2E_01459] [PRS_E2E_01460] [PRS_E2E_01461] [PRS_E2E_01462] [PRS_E2E_01463] [PRS_E2E_01464] [PRS_E2E_01466] [PRS_E2E_01467] [PRS_E2E_01468] [PRS_E2E_01469]
[RS_E2E_08529]	Each E2E profile shall use an appropriate subset of specific protection mechanisms	[PRS_E2E_00070] [PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00707] [PRS_E2E_00736] [PRS_E2E_00740] [PRS_E2E_00783] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00899] [PRS_E2E_00901] [PRS_E2E_00903] [PRS_E2E_00904] [PRS_E2E_00906] [PRS_E2E_00908] [PRS_E2E_00910] [PRS_E2E_01107] [PRS_E2E_01155] [PRS_E2E_01318]

▽

△

Requirement	Description	Satisfied by
[RS_E2E_08530]	Each E2E profile shall define a set of protection mechanisms and its behavior	[PRS_E2E_00196] [PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00707] [PRS_E2E_00736] [PRS_E2E_00740] [PRS_E2E_00783] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00899] [PRS_E2E_00901] [PRS_E2E_00903] [PRS_E2E_00904] [PRS_E2E_00906] [PRS_E2E_00908] [PRS_E2E_00910] [PRS_E2E_01107] [PRS_E2E_01155] [PRS_E2E_01318]
[RS_E2E_08531]	E2E Library shall call the CRC routines of CRC library	[PRS_E2E_00082] [PRS_E2E_00125] [PRS_E2E_00126] [PRS_E2E_00134] [PRS_E2E_00401] [PRS_E2E_00421] [PRS_E2E_00527] [PRS_E2E_00613] [PRS_E2E_01207] [PRS_E2E_01402]
[RS_E2E_08533]	CRC used in a E2E profile shall be different than the CRC used by the underlying physical communication protocol	[PRS_E2E_00070] [PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00707] [PRS_E2E_00736] [PRS_E2E_00740] [PRS_E2E_00783] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00899] [PRS_E2E_00901] [PRS_E2E_00903] [PRS_E2E_00904] [PRS_E2E_00906] [PRS_E2E_00908] [PRS_E2E_00910] [PRS_E2E_01107] [PRS_E2E_01155] [PRS_E2E_01318]
[RS_E2E_08534]	E2E protocol shall provide E2E Check status to the application	[PRS_E2E_00318] [PRS_E2E_00319] [PRS_E2E_00320] [PRS_E2E_00322] [PRS_E2E_00323] [PRS_E2E_00324] [PRS_E2E_00677] [PRS_E2E_00678] [PRS_E2E_00828] [PRS_E2E_00853] [PRS_E2E_00921] [PRS_E2E_00922] [PRS_E2E_00923] [PRS_E2E_00924] [PRS_E2E_00925] [PRS_E2E_01443]
[RS_E2E_08537]	SW-Cs shall tolerate a number of invalid/corrupted received data elements	[PRS_E2E_00646] [PRS_E2E_00648] [PRS_E2E_00651] [PRS_E2E_00654] [PRS_E2E_00657] [PRS_E2E_00660] [PRS_E2E_00663] [PRS_E2E_00666] [PRS_E2E_00706] [PRS_E2E_00735] [PRS_E2E_00851] [PRS_E2E_00852] [PRS_E2E_00869] [PRS_E2E_00872] [PRS_E2E_00876] [PRS_E2E_00881] [PRS_E2E_00886] [PRS_E2E_00887] [PRS_E2E_00893] [PRS_E2E_00898] [PRS_E2E_00900] [PRS_E2E_00902] [PRS_E2E_00905] [PRS_E2E_00907] [PRS_E2E_01415] [PRS_E2E_01416]

▽

△

Requirement	Description	Satisfied by
[RS_E2E_08539]	An E2E protection mechanism for inter-ECU communication of short to large data shall be provided	[PRS_E2E_00345] [PRS_E2E_00354] [PRS_E2E_00375] [PRS_E2E_00397] [PRS_E2E_00399] [PRS_E2E_00400] [PRS_E2E_00401] [PRS_E2E_00403] [PRS_E2E_00404] [PRS_E2E_00405] [PRS_E2E_00406] [PRS_E2E_00407] [PRS_E2E_00409] [PRS_E2E_00411] [PRS_E2E_00412] [PRS_E2E_00413] [PRS_E2E_00414] [PRS_E2E_00416] [PRS_E2E_00417] [PRS_E2E_00419] [PRS_E2E_00420] [PRS_E2E_00421] [PRS_E2E_00423] [PRS_E2E_00424] [PRS_E2E_00425] [PRS_E2E_00426] [PRS_E2E_00427] [PRS_E2E_00428] [PRS_E2E_00429] [PRS_E2E_00430] [PRS_E2E_00431] [PRS_E2E_00432] [PRS_E2E_00433] [PRS_E2E_00434] [PRS_E2E_00436] [PRS_E2E_00466] [PRS_E2E_00467] [PRS_E2E_00469] [PRS_E2E_00470] [PRS_E2E_00504] [PRS_E2E_00505] [PRS_E2E_00506] [PRS_E2E_00507] [PRS_E2E_00508] [PRS_E2E_00509] [PRS_E2E_00510] [PRS_E2E_00511] [PRS_E2E_00512] [PRS_E2E_00513] [PRS_E2E_00514] [PRS_E2E_00515] [PRS_E2E_00516] [PRS_E2E_00517] [PRS_E2E_00518] [PRS_E2E_00519] [PRS_E2E_00521] [PRS_E2E_00523] [PRS_E2E_00524] [PRS_E2E_00525] [PRS_E2E_00526] [PRS_E2E_00527] [PRS_E2E_00528] [PRS_E2E_00529] [PRS_E2E_00530] [PRS_E2E_00531] [PRS_E2E_00532] [PRS_E2E_00533] [PRS_E2E_00534] [PRS_E2E_00535] [PRS_E2E_00536] [PRS_E2E_00537] [PRS_E2E_00539] [PRS_E2E_00582] [PRS_E2E_00583] [PRS_E2E_00607] [PRS_E2E_00619] [PRS_E2E_00620] [PRS_E2E_00621] [PRS_E2E_00622] [PRS_E2E_00623] [PRS_E2E_00624] [PRS_E2E_00625] [PRS_E2E_00630] [PRS_E2E_00631] [PRS_E2E_00632] [PRS_E2E_00633] [PRS_E2E_00634] [PRS_E2E_00635] [PRS_E2E_00636] [PRS_E2E_00637] [PRS_E2E_00639] [PRS_E2E_00640] [PRS_E2E_00654] [PRS_E2E_00675] [PRS_E2E_00676] [PRS_E2E_00881] [PRS_E2E_01156] [PRS_E2E_01157] [PRS_E2E_01159] [PRS_E2E_01161] [PRS_E2E_01162] [PRS_E2E_01163] [PRS_E2E_01164] [PRS_E2E_01165] [PRS_E2E_01166] [PRS_E2E_01167] [PRS_E2E_01169] [PRS_E2E_01170] [PRS_E2E_01171] [PRS_E2E_01172] [PRS_E2E_01173] [PRS_E2E_01174] [PRS_E2E_01175] [PRS_E2E_01176] [PRS_E2E_01177] [PRS_E2E_01178] [PRS_E2E_01179] [PRS_E2E_01180] [PRS_E2E_01181] [PRS_E2E_01182] [PRS_E2E_01183] [PRS_E2E_01184] [PRS_E2E_01185] [PRS_E2E_01186] [PRS_E2E_01187] [PRS_E2E_01188] ▽

▽

△

Requirement	Description	Satisfied by
		△ [PRS_E2E_01189] [PRS_E2E_01190] [PRS_E2E_01191] [PRS_E2E_01192] [PRS_E2E_01193] [PRS_E2E_01194] [PRS_E2E_01195] [PRS_E2E_01196] [PRS_E2E_01197] [PRS_E2E_01198] [PRS_E2E_01203] [PRS_E2E_01205] [PRS_E2E_01206] [PRS_E2E_01209] [PRS_E2E_01210] [PRS_E2E_01211] [PRS_E2E_01212] [PRS_E2E_01213] [PRS_E2E_01214] [PRS_E2E_01215] [PRS_E2E_01216] [PRS_E2E_01217] [PRS_E2E_01218] [PRS_E2E_01219] [PRS_E2E_01220] [PRS_E2E_01221] [PRS_E2E_01222] [PRS_E2E_01223] [PRS_E2E_01224] [PRS_E2E_01225] [PRS_E2E_01226] [PRS_E2E_01227] [PRS_E2E_01228] [PRS_E2E_01253] [PRS_E2E_01254] [PRS_E2E_01255] [PRS_E2E_01400] [PRS_E2E_01401] [PRS_E2E_01402] [PRS_E2E_01403] [PRS_E2E_01405] [PRS_E2E_01406] [PRS_E2E_01407] [PRS_E2E_01408] [PRS_E2E_01415] [PRS_E2E_01416] [PRS_E2E_01420] [PRS_E2E_01422] [PRS_E2E_01424] [PRS_E2E_01427] [PRS_E2E_01429] [PRS_E2E_01430] [PRS_E2E_01433] [PRS_E2E_01437] [PRS_E2E_01438] [PRS_E2E_01439] [PRS_E2E_01440] [PRS_E2E_01441] [PRS_E2E_01442] [PRS_E2E_01471] [PRS_E2E_01472]
[RS_E2E_08540]	E2E protocol shall support protected periodic/mixed periodic communication	[PRS_E2E_01444] [PRS_E2E_01445] [PRS_E2E_01446]
[RS_E2E_08541]	E2E protocol shall support protected non-periodic communication	[PRS_E2E_01444] [PRS_E2E_01446] [PRS_E2E_01465]
[RS_E2E_08542]	E2E protocol shall support dynamic restart of communication peers	[PRS_E2E_00324] [PRS_E2E_00923] [PRS_E2E_00924] [PRS_E2E_00925]
[RS_E2E_08543]	E2E protocol shall support variable length of transmitted data	[PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_01318]
[RS_E2E_08544]	E2E protocol shall provide a timeout detection mechanism	[PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_01318]

▽

△

Requirement	Description	Satisfied by
[RS_E2E_08545]	E2E protocol shall provide a detection mechanism for corrupted data	[PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_01318]
[RS_E2E_08546]	E2E protocol shall provide a detection mechanism for masquerade or incorrect addressing	[PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_01318]
[RS_E2E_08547]	E2E protocol shall provide a detection mechanism for repetition, insertion or incorrect sequence of data	[PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_01318]
[RS_E2E_08548]	E2E protocol shall provide E2E overall state to the application	[PRS_E2E_00218] [PRS_E2E_00219] [PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00480] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00678] [PRS_E2E_00865] [PRS_E2E_00870] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00904] [PRS_E2E_00922] [PRS_E2E_01318]
[RS_E2E_08549]	Each E2E profile shall have a unique Profile ID	[PRS_E2E_00372] [PRS_E2E_00394] [PRS_E2E_00479] [PRS_E2E_00503] [PRS_E2E_00522] [PRS_E2E_00736] [PRS_E2E_00875] [PRS_E2E_00877] [PRS_E2E_00882] [PRS_E2E_00888] [PRS_E2E_00894] [PRS_E2E_00901] [PRS_E2E_01318]
[RS_E2E_08550]	The implementation of the E2E Supervision shall provide at least one of the E2E Profiles	[PRS_E2E_00372] [PRS_E2E_00875] [PRS_E2E_01318]

Table 5.1: Requirements Tracing

6 Functional specification

This chapter contains the specification of the internal functional behavior of the E2E supervision, this includes how the layout of the E2E-Header is defined, how the E2E-Header is created and how the E2E-Header is evaluated, and how the E2E-Statemachine is defined. For general introduction of the E2E supervision, see Chapter 1.

6.1 Overview of communication protection

An important aspect of a communication protection mechanism is its standardization and its flexibility for different purposes. This is resolved by having a set of E2E Profiles, that define a combination of protection mechanisms, a message format, and a set of configuration parameters.

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

E2E communication protection works as follows:

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

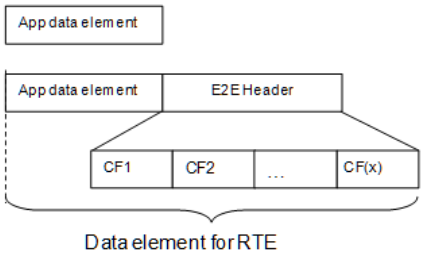

Figure 6.1: Safety protocol concept (with exemplary location of the E2E header)

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

6.2 Overview of E2E Profiles

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

Each E2E Profile provides an alternative way to protect the communication, by means of different algorithms. However, E2E Profiles have similar interfaces and behavior.

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

1. A CRC, provided by CRC Supervision;
2. A Sequence Counter incremented at every transmission request, the value is checked at receiver side for correct incrementation;
3. An Alive Counter incremented at every transmission request, the value checked at the receiver side if it changes at all, but correct incrementation is not checked;
4. A specific ID for every port data element sent over a port or a specific ID for every message-group (global to system, where the system may contain potentially several ECUs);
5. A specific ID for every source (e.g., client) of a data element or message group
6. A message type distinguishing between requests and responses in case of E2E communication protection for methods
7. A message result distinguishing between normal and error responses in case of E2E communication protection for methods
8. Timeout detection:
   (a) Receiver communication timeout.
   (b) Sender acknowledgement timeout.

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

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

The E2E Profiles can be used for both inter and intra ECU communication. The E2E Profiles were specified for specific communication infrastructure, such as CAN, CAN FD, FlexRay, LIN, Ethernet.

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

6.2.1 Error detection

[PRS_E2E_00012]

Upstream requirements: RS_E2E_08528

⌈The internal Supervision mechanisms error detection and reporting shall be implemented according to the pre-defined E2E Profiles specified in this document.⌋

[PRS_E2E_00673]

Upstream requirements: RS_E2E_08528

⌈The AUTOSAR E2E-Protocol shall use the error codes defined in [PRS_E2E_00855].⌋

[PRS_E2E_00855] Error Codes

Upstream requirements: RS_E2E_08528

⌈

Type or error or status	Related code
At least one pointer parameter is a NULL pointer	E2E_E_INPUTERR_NULL
At least one input parameter is erroneous, e.g. out of range	E2E_E_INPUTERR_WRONG
Function completed successfully	E2E_E_OK

⌋

6.2.2 Common Types of E2E Profiles

Some E2E profile make use of common data types which are shared among the different profiles. – Those shared types are introduced in this section instead of the profile specific sections.

6.2.2.1 Profile Xm Message Type Enumeration

[PRS_E2E_00739]

Upstream requirements: RS_E2E_08528

⌈The MessageType argument of the E2E_PXXmProtect, E2E_PXXmForward, and E2E_PXXmCheck functions shall be set to one of the following enumeration values (see [PRS_E2E_00856]).⌋

[PRS_E2E_00856] E2E Profile Xm Message Type Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	Value	Description

STD_MESSAGETYPE_REQUEST	0	The type of the message is a request message which is sent from the client to the server.
STD_MESSAGETYPE_RESPONSE	1	The type of the message is a response message which is sent from the server to the client.

⌋

6.2.2.2 Profile Xm Message Result Enumeration

[PRS_E2E_00743]

Upstream requirements: RS_E2E_08528

⌈The MessageResult argument of the E2E_PXXmProtect, E2E_PXXmForward, and E2E_PXXmCheck functions shall be set to one of the following enumeration values (see [PRS_E2E_00857]).⌋

[PRS_E2E_00857] E2E Profile Xm Message Result Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	Value	Description
STD_MESSAGERESULT_OK	0	The type of the result in the response message is a normal (i.e., a non erroneus) result. This value is also used for the request messages, where the value of this field is fixed to this value.
STD_MESSAGERESULT_ERROR	1	The type of the result int the response message is an error (i.e., an erroneus) result.

⌋

6.2.3 General Functionality of an E2E-Profile

Each E2E-Profile provides the following 3 functionalities:

1. Protect
2. Forward
3. Check

The 'protect' functionality, simply called the 'protect function' creates the E2E-Header and therefore protects the data to be sent over a communication medium.

The 'forward' functionality, simply called the 'forward function', is similar to the protect function and creates the header for the data to be transmitted but allows the additional replication of a received E2E-State. The main use-case for this function is Signal-Service-Translation where e.g. a E2E-protected signal is received, and the E2E-Status shall be replicated on the outgoing side.

The 'check' functionality, simply called the 'check function', evaluates the E2E-Header of the received message and checks for occurred communication faults. These faults are mirrored in the returned E2E-States.

In addition to the single E2E-Profiles a E2E-Statemachine evaluates the returned E2E-States over a longer period.

6.2.3.1 Functionality of the Counter

On the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function,

   b. the data is repeated

2. OK:

   a. counter is incremented by one (i.e. no data lost),

   b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Error: a. counter is incremented more than allowed (i.e. too many data lost).

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

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

6.2.3.2 Timeout detection

The previously mentioned mechanisms (e.g. for Profile 5: CRC, Counter, Data ID) enable to check the validity of received data element, when the receiver is running independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively messages, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

The attribute State->NewDataAvailable == FALSE means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_PXXSTATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

6.2.3.3 Cyclic Redundancy Check

The Cyclic Redundancy Check, short CRC, is used to determine if bits flipped during the transmission of a message.

In contrast to errors indicated based on the evaluation of the counter - CRC-errors are unlikely to be a ‚false alarm' (e.g. when using a good CRC-polynomial a detected CRC-error indicates that a data corruption occurred). Considering this fact, it is implausible that a stream of data without any detected CRC-errors contains a significant number of undetected corrupted data.

Due to this, a more stringent reaction upon CRC-errors is adequate. After detection of the first CRC-error on the subsequent data stream may contain a significant number of undetected corrupted data.

The maximum number of CRC-errors a receiver tolerates shall be limited, because the probability of receiving more than one undetected erroneous, within its error detection and qualification time interval, messages cannot be neglected. A wrong CRC indicates, that the integrity of the communication channel is affected.

The fault tolerance designed into the receiver (see UC_E2E_00170) may be exceeded as a possible consequence.

6.3 Specification of E2E Profiles - Generalized Part

This chapter contains the part of the specification for E2E profiles that is used in more than one profile specification. The behavior of E2E profiles is described independently of a specific profile. Text and figures use placeholder like ”XX” which are replaced by a profile-specific value or text. All profile-specific content, including these placeholders, is defined in the corresponding profile-specific sub-chapter. This chapter does not apply to method profiles. This chapter does only apply to profiles where the fields of DataID and Length are part of the profile header. The E2E mechanisms can detect the following faults or effects of faults:

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC

Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 6.1: Detectable communication faults

6.3.1 Counter

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

[PRS_E2E_01205]

Upstream requirements: RS_E2E_08539

⌈In E2E Profiles, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value, then it shall restart with 0 for the next send request. The maximum value of the counter depends on the size of the counter It is 0xFF (8bit counter), 0xFF'FF (16bit counter) or 0xFF'FF'FF'FF (32 bit).⌋

6.3.2 Data ID

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

[PRS_E2E_01438] Uniqueness of Data IDs

Replaces: PRS_E2E_UC_01204

Upstream requirements: RS_E2E_08539

⌈In E2E profiles, the Data IDs should be globally unique within the network of communicating system (made of several ECUs each sending different data).⌋

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

In case of usage of E2E Supervision for protecting messages (i.e. invocation from COM), the receiver COM expects at a reception only a specific message, which is checked by E2E Supervision using Data ID.

6.3.3 Length

The Length field is introduced to support variable-size length - the Data [] array storing the serialized data can potentially have a different length in each cycle. The Length includes user data + E2E Header (CRC + Counter + Length + DataID).

6.3.4 CRC

E2E Profiles use CRC of different length depending on the length of the message to ensure a high detection rate and high Hamming Distance.

[PRS_E2E_01206]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E Profiles shall use the CRC functions listed in chapters Chapter 6.7, Chapter 6.10, Chapter 6.11, Chapter 6.14, for calculating the CRC.⌋

Note: The Crc_CalculateCRC32P4() is different from the 32 bit CRCs used by FlexRay, CAN and TCP/IP. It is also provided by different software modules (FlexRay, CAN and TCP/IP stack CRCs/checksums are provided by hardware support in Communication Controllers or by communication stack software, but not by CRC Supervision).

[PRS_E2E_01207]

Upstream requirements: RS_E2E_08531

⌈The CRC shall be calculated over the entire E2E header (excluding the CRC bytes) and over the user data.⌋

6.3.5 Timeout detection

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

6.3.6 Creation of E2E-Header

6.3.6.1 E2E_PXXProtect()

The function E2E_PXXProtect() performs the steps as specified by the following diagrams in this section.

[PRS_E2E_01209]

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXProtect() shall have the overall behavior as shown in Figure 6.2.⌋

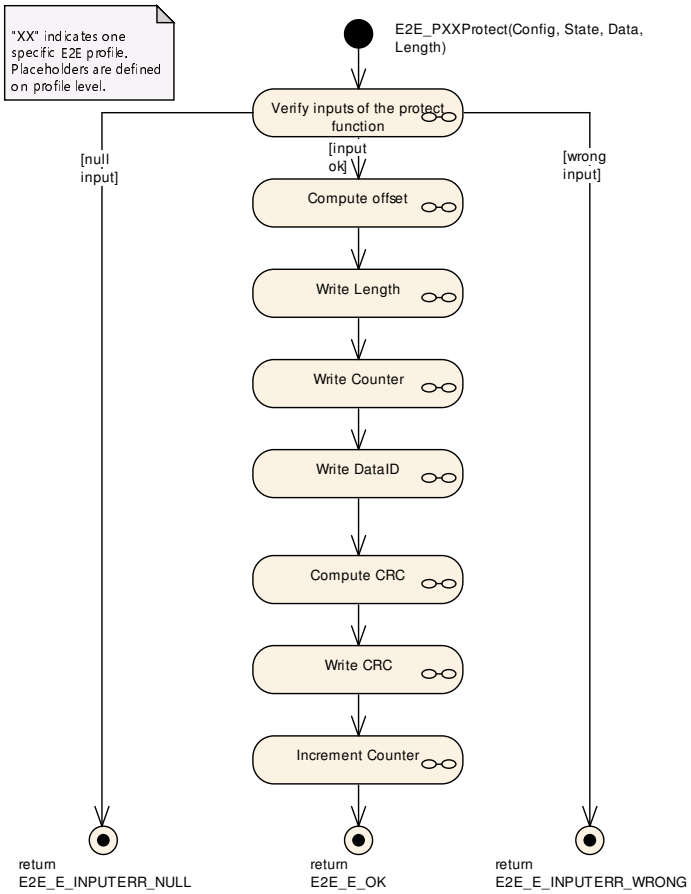

Figure 6.2: Behaviour of E2E_PXXProtect()

[PRS_E2E_01210]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_PXXProtect() shall behave as shown in Figure 6.3.⌋

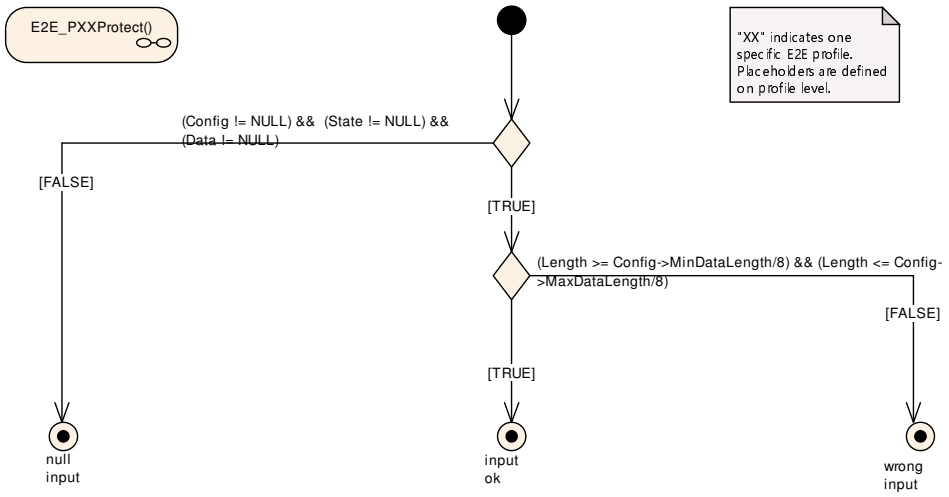

Figure 6.3: E2E_PXXProtect() step ”Verify inputs of the protect function”

[PRS_E2E_01211]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute offset” in E2E_PXXProtect(), E2E_PXXForward() and E2E_PXXCheck() shall behave as shown in Figure 6.4.⌋

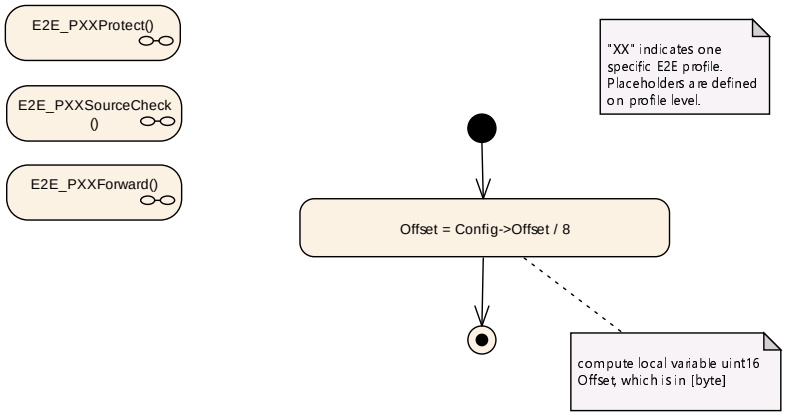

Figure 6.4: E2E_PXXProtect(), E2E_PXXForward() and E2E_PXXCheck() step ”Compute offset”

[PRS_E2E_01212]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Length” in E2E_PXXProtect() and E2E_PXXForward() shall behave as shown in Figure 6.5.⌋

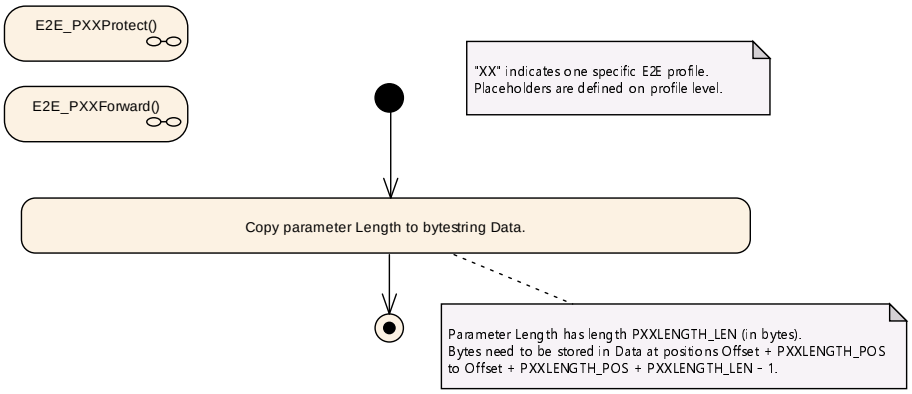

Figure 6.5: E2E_PXXProtect() and E2E_PXXForward() step ”Write Length”

[PRS_E2E_01213]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_PXXProtect() shall behave as shown in Figure 6.6.⌋

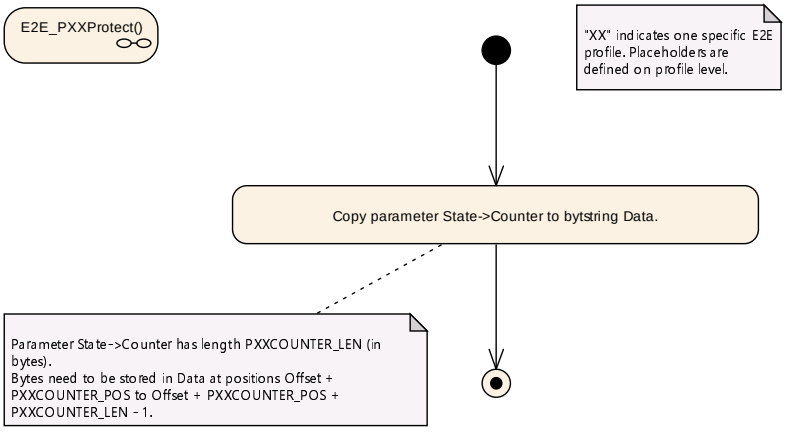

Figure 6.6: E2E_PXXProtect() step ”Write Counter”

[PRS_E2E_01214]

Upstream requirements: RS_E2E_08539

⌈The step ”Write DataID” in E2E_PXXProtect() shall behave as shown in Figure 6.7.⌋

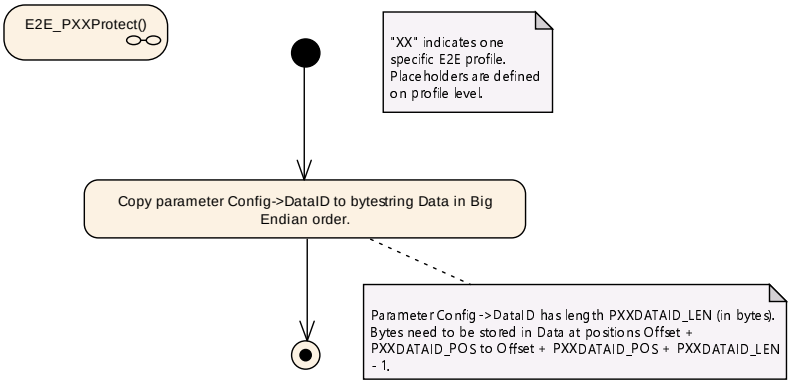

Figure 6.7: E2E_PXXProtect() step ”Write DataID”

[PRS_E2E_01215]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_PXXProtect(), E2E_PXXForward() and in E2E_PXXCheck() shall behave as shown in Figure 6.8.⌋

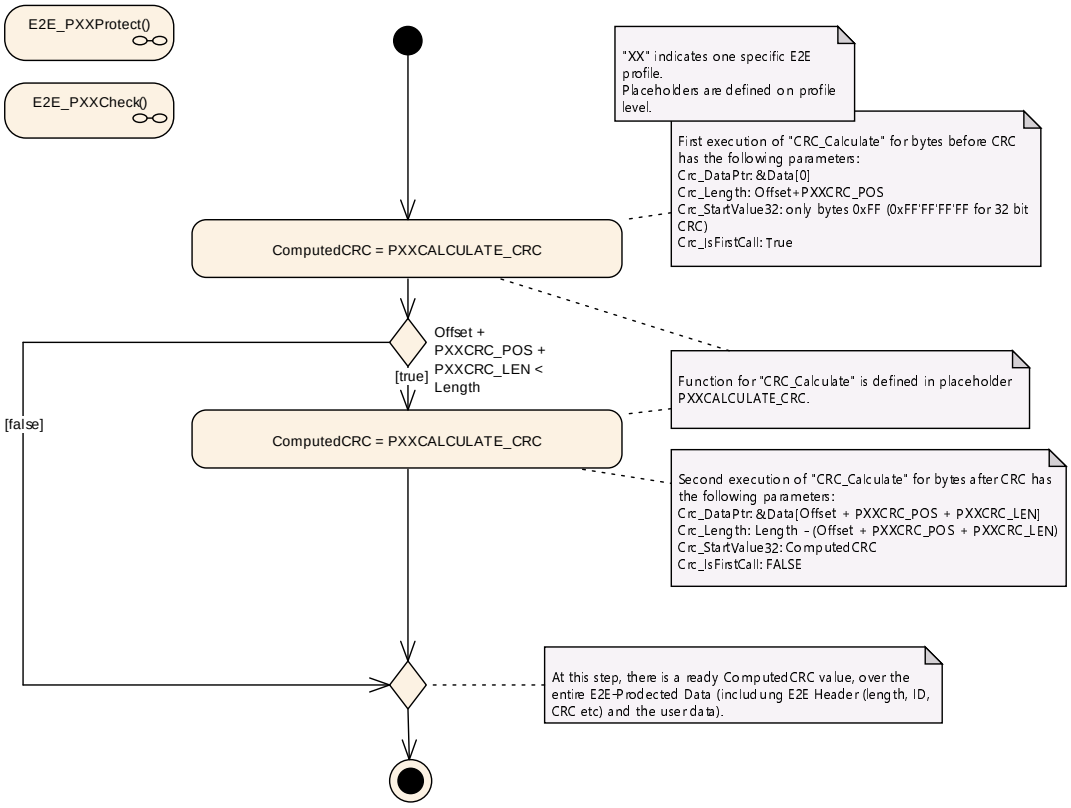

Figure 6.8: E2E_PXXProtect(), E2E_PXXCheck() and E2E_PXXForward() step ”Compute CRC”

[PRS_E2E_01216]

Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC” in E2E_PXXProtect() and E2E_PXXForward() shall behave as shown in Figure 6.9.⌋

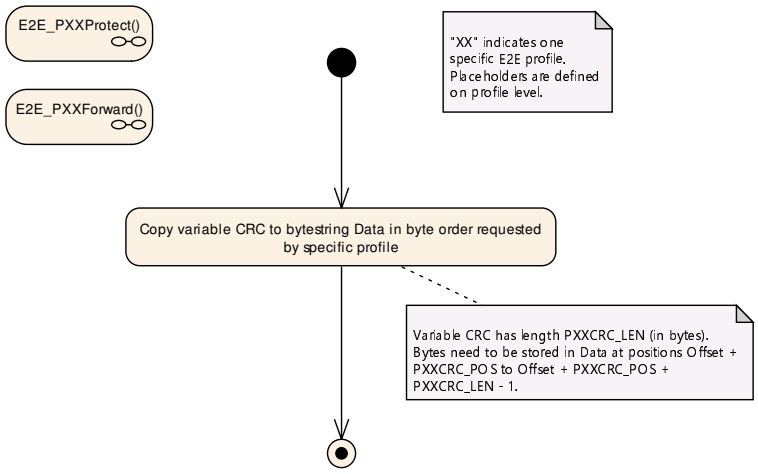

Figure 6.9: E2E_PXXProtect() and E2E_PXXForward() step ”Write CRC”

[PRS_E2E_01217]

Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_PXXProtect() and E2E_PXXForward() shall behave as shown in Figure 6.10.⌋

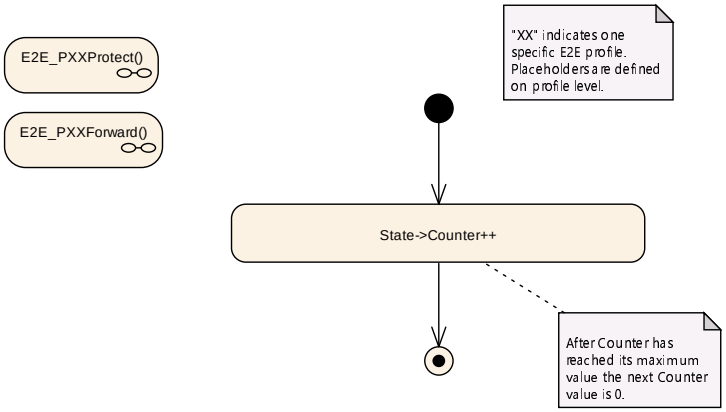

Figure 6.10: E2E_PXXProtect() and E2E_PXXForward() step ”Increment Counter”

6.3.6.2 E2E_PXXForward()

The E2E_PXXForward() function of an E2E profile is called by a SW-C to protect its application data and to forward an received E2E status for use cases like translation of signal based to service oriented communication. If the received E2E status equals E2E_P_OK the behavior of the function shall be the same like E2E_PXXProtect(). The function E2E_PXXForward() performs the steps as specified by the following diagrams in this section.

[PRS_E2E_01218]

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXForward() shall have the overall behavior as shown in Figure 6.11.⌋

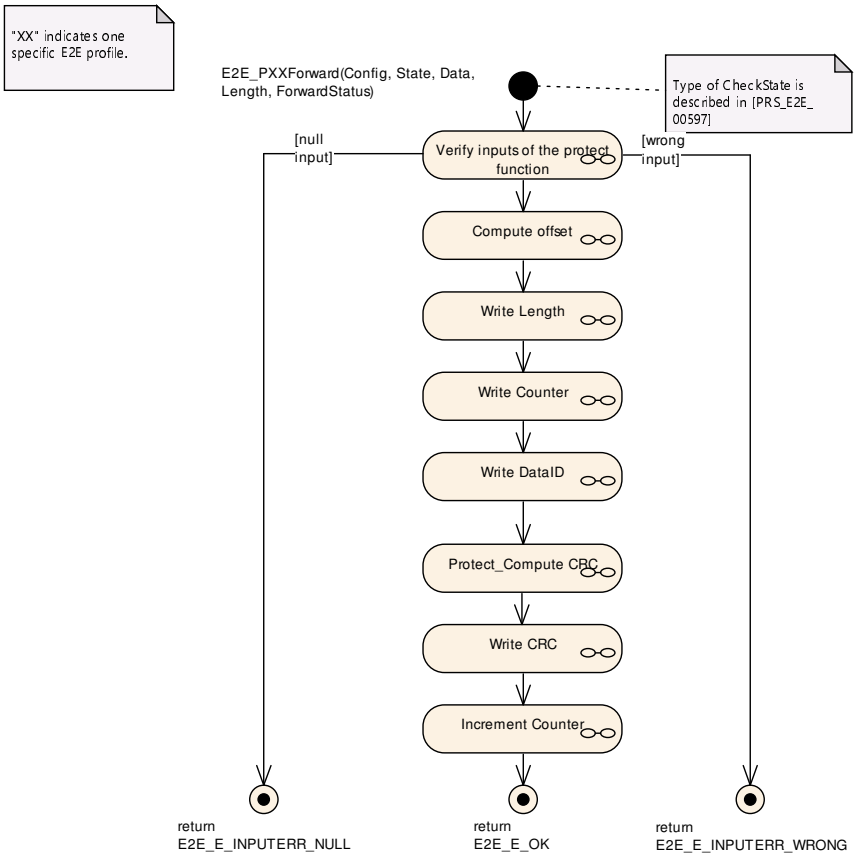

Figure 6.11: Behaviour of E2E_PXXForward()

Following steps are described in section 6.3.6.1

- ”Compute Offset” see [PRS_E2E_01211]
- ”Write Length” see [PRS_E2E_01212]
- ”Compute CRC” see [PRS_E2E_01215]
- ”Write CRC” see [PRS_E2E_01216]
- ”Increment Counter” see [PRS_E2E_01217]

[PRS_E2E_01228]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the forward function” in E2E_PXXForward() shall behave as shown in Figure 6.12.⌋

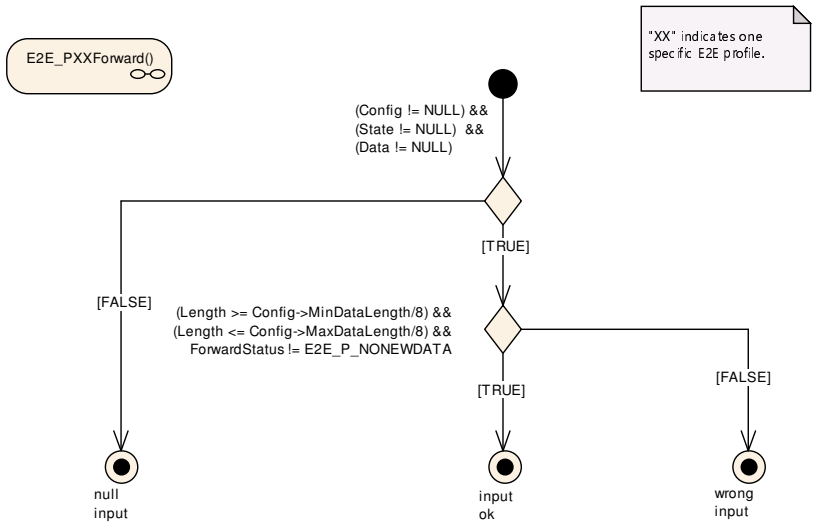

Figure 6.12: E2E_PXXForward() step ”Verify inputs of the forward function”

[PRS_E2E_01219]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_PXXForward() shall behave as shown in Figure 6.13.⌋

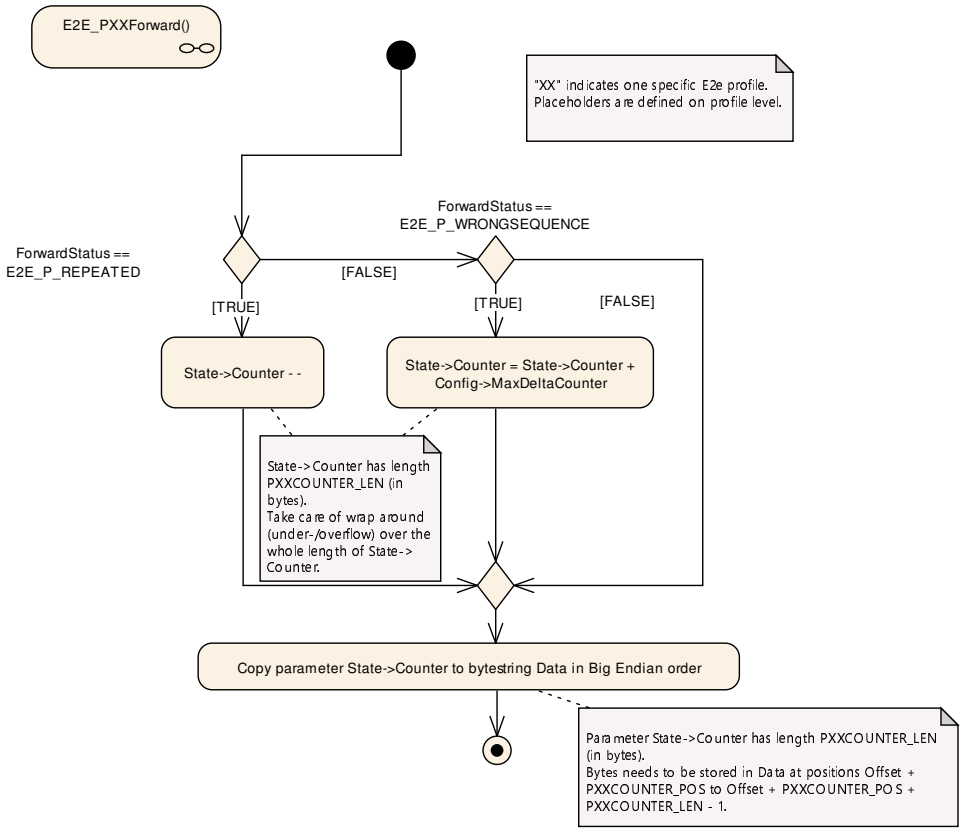

Figure 6.13: E2E_PXXForward() step ”Write Counter”

[PRS_E2E_01220]

Upstream requirements: RS_E2E_08539

⌈The step ”Write DataID” in E2E_PXXForward() shall behave as shown in Figure 6.14.⌋

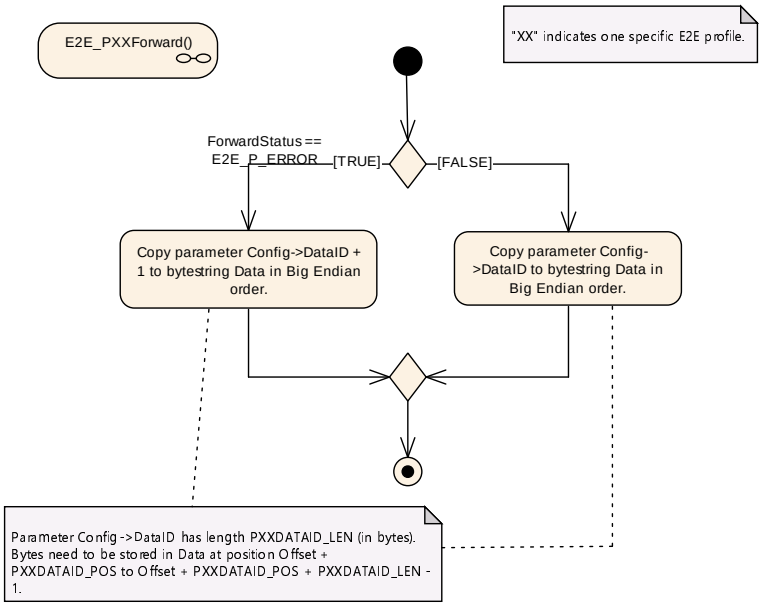

Figure 6.14: E2E_PXXForward() step ”Write DataID”

6.3.7 Evaluation of the E2E-Header

6.3.7.1 E2E_PXXCheck()

The function E2E_PXXCheck() performs the actions as specified by the following diagrams in this section and according to diagram Figure 6.15.

[PRS_E2E_01221]

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXCheck() shall have the overall behavior as shown in Figure 6.15.⌋

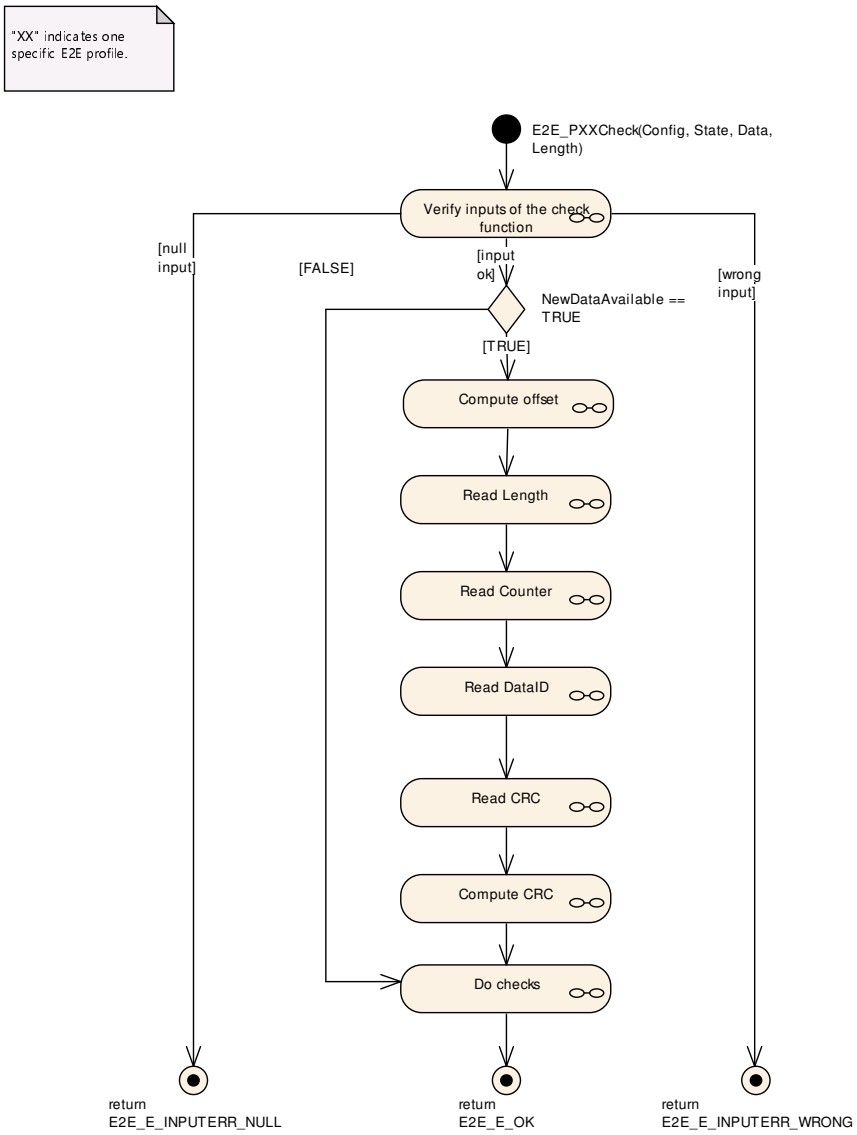

Figure 6.15: Behaviour of E2E_PXXCheck()

Following steps are described in section 6.3.6.1

- ”Compute Offset” see [PRS_E2E_01211]
- ”Compute CRC” see [PRS_E2E_01215]

[PRS_E2E_01222]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the check function” in E2E_PXXCheck() shall behave as shown in Figure 6.16.⌋

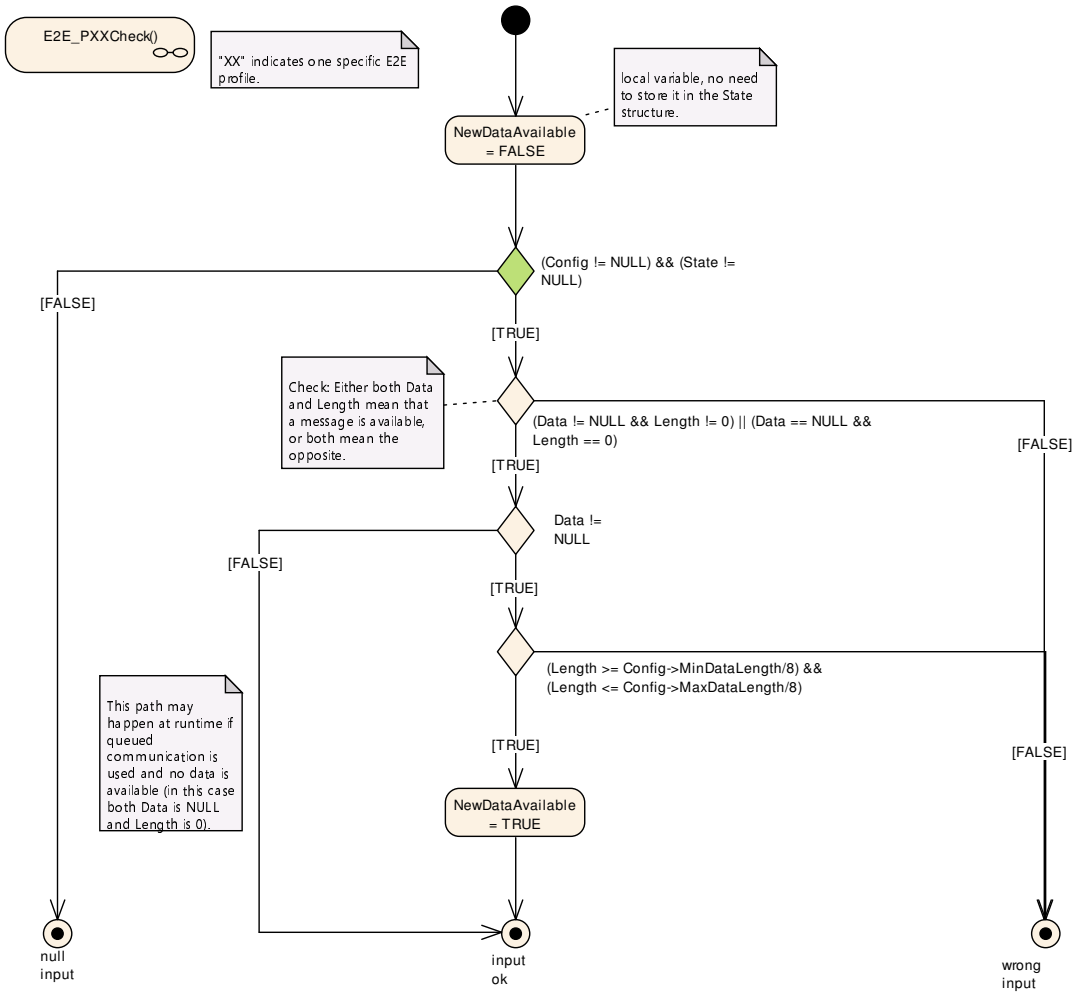

Figure 6.16: E2E_PXXCheck() step 'Verify inputs of the check function"

[PRS_E2E_01223]

Upstream requirements: RS_E2E_08539

⌈The step "Read Length" in E2E_PXXCheck() shall behave as shown in Figure 6.17.⌋

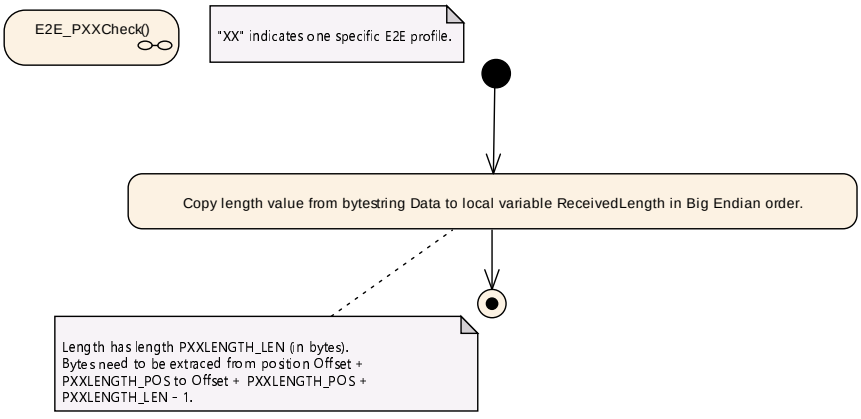

Figure 6.17: E2E_PXXCheck() step "Read Length"

[PRS_E2E_01224]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Counter” in E2E_PXXCheck() shall behave as shown in Figure 6.18.⌋

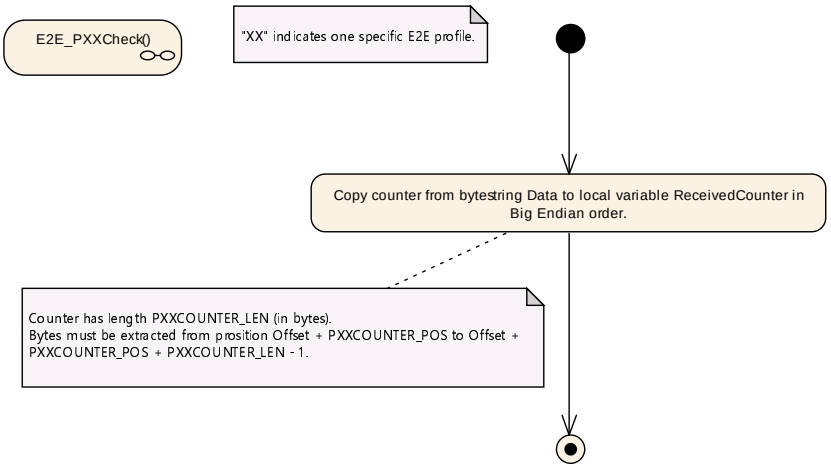

Figure 6.18: E2E_PXXCheck() step ”Read Counter”

[PRS_E2E_01225]

Upstream requirements: RS_E2E_08539

⌈The step ”Read DataID” in E2E_PXXCheck() shall behave as shown in Figure 6.19.⌋

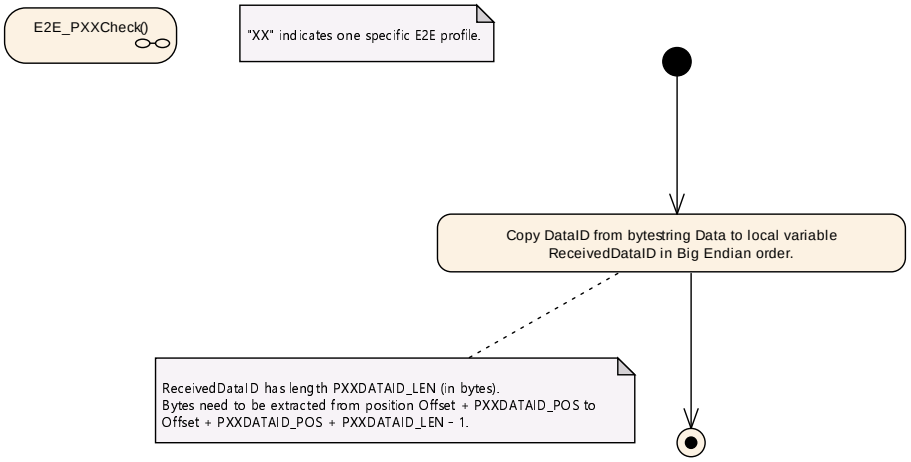

Figure 6.19: E2E_PXXCheck() step ”Read DataID”

[PRS_E2E_01226]

Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_PXXCheck() shall behave as shown in Figure 6.20.⌋

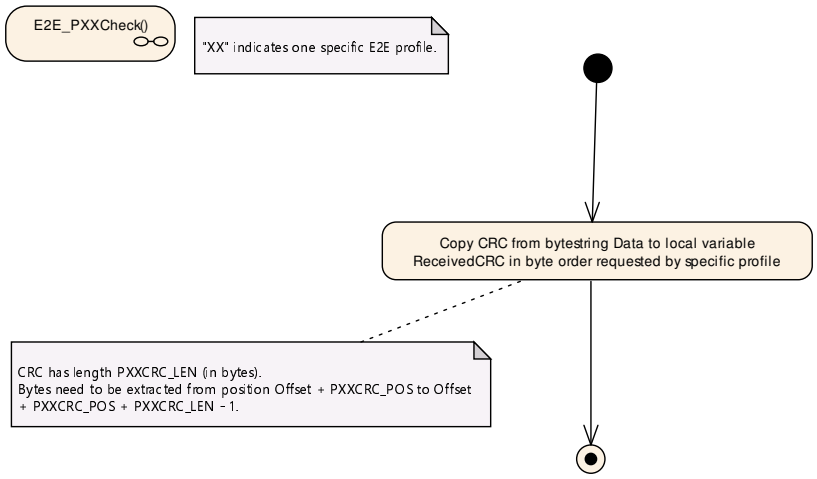

Figure 6.20: E2E_PXXCheck() step ”Read CRC”

[PRS_E2E_01227]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks” in E2E_PXXCheck() shall behave as shown in Figure 6.21.⌋

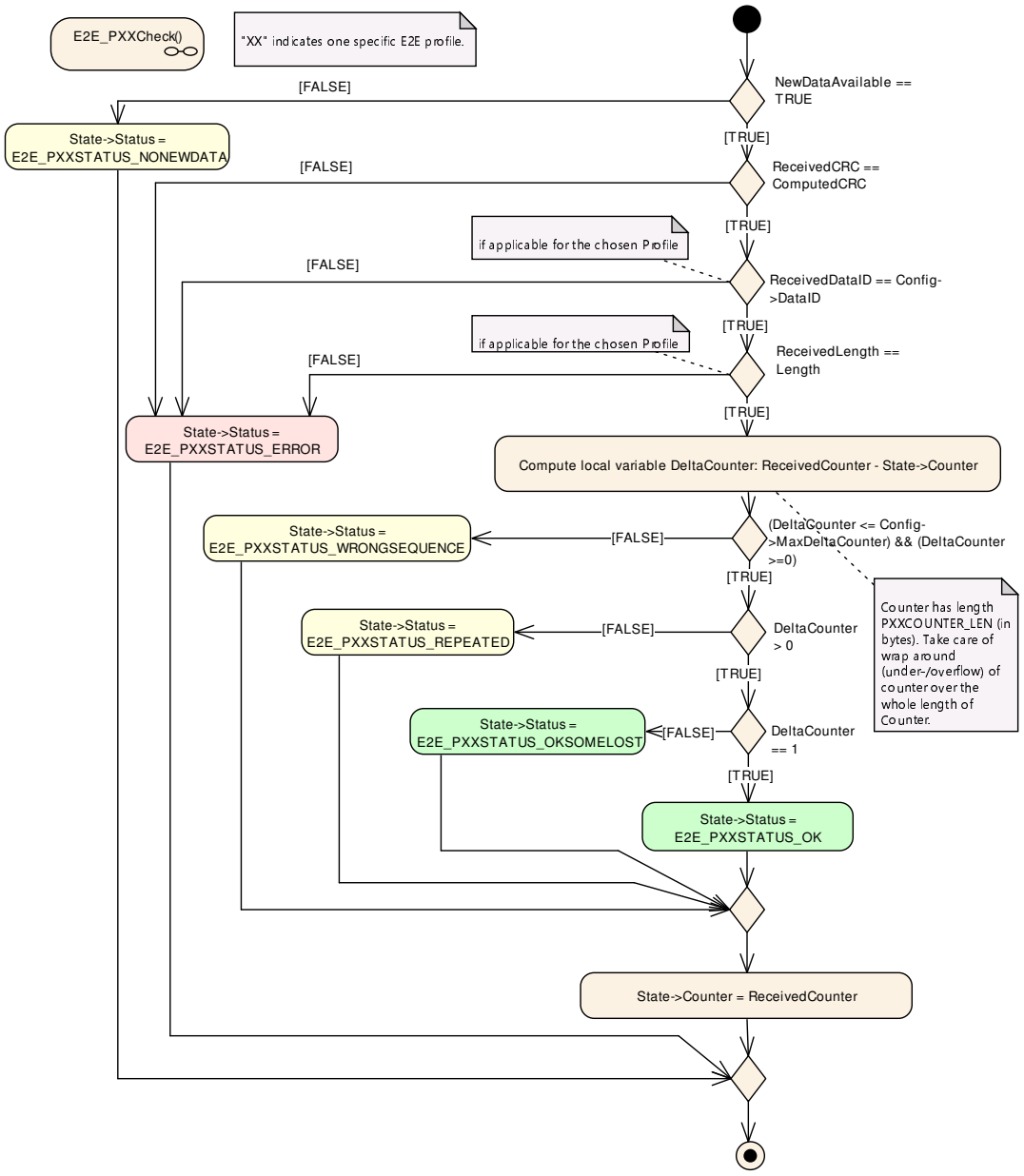

Figure 6.21: E2E_PXXCheck() step ”Do Checks”

6.3.8 Get Header Info

[PRS_E2E_01471] PXX_GetHeaderInfo

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXGetHeaderInfo() shall behave as shown in Figure 6.22.⌋

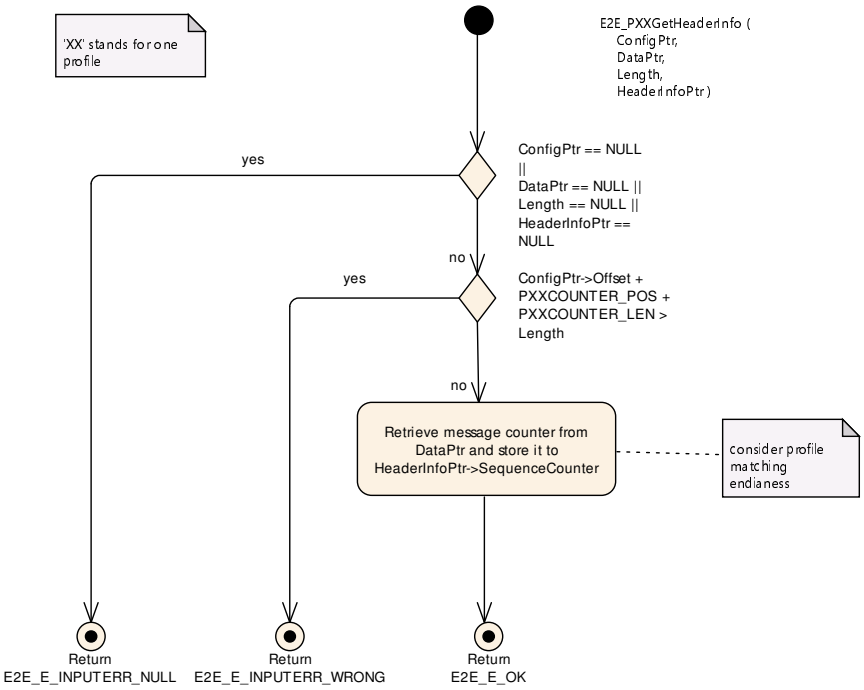

Figure 6.22: PXX_GetHeaderInfo

6.3.9 Profile Data Types

6.3.9.1 Profile Protect State Type

[PRS_E2E_01250]

Upstream requirements: RS_E2E_08528

⌈The E2E_PXXProtect and E2E_PXXForward functions 'state' shall have the members defined in [PRS_E2E_00858].⌋

[PRS_E2E_00858] E2E Profile Protect State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
Counter	Unsigned Integer	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_PXXProtect() is called, it increments the counter up to the maximum value (0xFF for 8 bit counter, 0xFF'FF for a 16 bit counter and 0xFF'FF'FF'FF for a 32 bit counter). After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.

⌋

6.3.9.2 Profile Check Status Type

[PRS_E2E_01251]

Upstream requirements: RS_E2E_08528

⌈The E2E_PXXCheck functions 'State' shall have the members defined in [PRS_E2E_00859].⌋

[PRS_E2E_00859] E2E Profile XX Check State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
Counter	Unsigned Integer	Counter of the data in previous cycle.
Status	Enumeration	Result of the verification of the Data in this cycle, determined by the Check function.

⌋

6.3.9.3 Profile Check Status Enumeration

[PRS_E2E_01252]

Upstream requirements: RS_E2E_08528

⌈The E2E_PXXCheck functions 'State->Status' enumeration type shall consist of the following enumeration values (see [PRS_E2E_00860]).⌋

[PRS_E2E_00860] E2E Profile Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_PXXSTATUS_OK	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
E2E_PXXSTATUS_NONEWDATA	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_PXXSTATUS_REPEATED.
E2E_PXXSTATUS_ERROR	Error	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).

E2E_PXXSTATUS_REPEATED	Error	The checks of the Data in this cycle were successful, with the exception of the repetition.
E2E_PXXSTATUS_OKSOMELOST	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
E2E_PXXSTATUS_WRONGSEQUENCE	Error	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta

⌋

6.3.9.4 GetHeaderInfo Data Types

[PRS_E2E_01466] Data Types of PXX_GetHeaderInfo

Upstream requirements: RS_E2E_08528

⌈The PXX_GetHeaderInfo type shall have the members defined in [PRS_E2E_01467].⌋

[PRS_E2E_01467] Data Types Description

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
SequenceCounter	Unsigned Integer	Counter to be retrieved from Data

⌋

6.4 Specification of E2E Profiles for Methods - Generalized Part

This chapter contains the part of the specification for E2E profiles for methods that is used in more than one profile specification. The behavior of E2E profiles is described independently of a specific profile. Text and figures use placeholder like ”XXm” which are replaced by a profile-specific value or text. All profile-specific content, including these placeholders, is defined in the corresponding profile-specific sub-chapter. This chapter applies to all method profiles where the fields of DataID and Length are part of the profile header.

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

Fault	Main safety mechanisms
Repetition of information	Counter

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

Table 6.2: Detectable communication faults using method profiles

6.4.1 Counter

In E2E profiles for methods, the counter is initialized, incremented, reset and checked by the E2E profile. The counter is not manipulated or used by the caller of the E2E supervision.

[PRS_E2E_01156]

Upstream requirements: RS_E2E_08539

⌈In E2E profiles for methods, the counter on the sender side shall be initialized with 0 for the first transmission request of a data element the counter and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF'FF for a 16bit counter, 0xFF'FF'FF'FF for a 32bit counter, 0xFF'FF'FF'FF' FF'FF'FF'FF for a 64bit counter), then it shall restart with 0 for the next send request.⌋

Note that the maximum counter value (0xFF'FF for a 16bit counter, 0xFF'FF'FF'FF for a 32bit counter, 0xFF'FF'FF'FF' FF'FF'FF'FF for a 64bit counter) is not reserved as a special invalid value, but it is used as a normal counter value.

In E2E profiles for methods, the following is detected on the receiver side by evaluating the counter of received data against the counter of previously received data:

1. Repetition:
   a. no new data has arrived since last invocation of E2E supervision check function,
   b. the data is repeated
2. OK:
   a. counter is incremented by one (i.e. no data lost),

b. counter is incremented by more than one, but still within allowed limits (i.e. some data lost),

Case 1 corresponds to the failed alive counter check, and case 2 correspond to successful alive counter check.

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

6.4.2 Data ID

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

[PRS_E2E_01157]

Upstream requirements: RS_E2E_08539

⌈In the E2E profiles for methods, the Data ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.⌋

[PRS_E2E_01439] Uniqueness of Data IDs

Replaces: PRS_E2E_UC_01158
Upstream requirements: RS_E2E_08539

⌈In the E2E profiles for methods, the Data IDs should be globally unique within the network of communicating systems (made of several ECUs each sending different data).⌋

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

In case of usage of E2E Supervision for protecting messages (i.e. invocation from COM), the receiver COM expects at a reception only a specific message, which is checked by E2E Supervision using Data ID.

6.4.3 Length

The Length field is introduced to support variable-size length - the Data [] array storing the serialized data can potentially have a different length in each cycle. The Length includes user data + E2E Header (CRC + Counter + Length + DataID).

6.4.4 CRC

E2E profiles for methods uses a suitable CRC, to ensure a high detection rate and high Hamming Distance.

[PRS_E2E_01159]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E profiles for methods shall use the function defined in PXXMCALCULATE_CRC of the SWS CRC supervision for calculating the CRC.⌋

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

[PRS_E2E_01160]

Upstream requirements: RS_E2E_08528

⌈In E2E profiles for methods, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes) and over the user data.⌋

6.4.5 Message Type

The Message Type field is used to distinguish request messages from response messages in method communication.

[PRS_E2E_01161]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In the E2E profiles for methods the Message Type field shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.⌋

[PRS_E2E_01162]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈A Message Type field with a value of 0 indicates a request message.⌋

[PRS_E2E_01163]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈A Message Type field with a value of 1 indicates a response message.⌋

6.4.6 Message Result

The Message Result field is used to distinguish normal response messages from error response messages in method communication.

[PRS_E2E_01203]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈In the E2E profiles for methods the Message Result field shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.⌋

[PRS_E2E_01164]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈A Message Result field with a value of 0 indicates a normal response message.⌋

[PRS_E2E_01165]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈A Message Result field with a value of 1 indicates an error response message.⌋

[PRS_E2E_01166]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈The Message Result field shall be set to 0 for request messages (i.e., in case the Message Type field is set to 0).⌋

6.4.7 Source ID

The unique Source IDs are to verify the identity of the source of each transmitted safety-related data element. In case of method communication, the Source ID identifies the client which performs a method call.

[PRS_E2E_01167]

Upstream requirements: RS_E2E_08539

⌈In the E2E profiles for methods, the Source ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.⌋

[PRS_E2E_01440] Uniqueness of Data IDs

Replaces: PRS_E2E_UC_01168
Upstream requirements: RS_E2E_08539

⌈In the E2E profiles for methods, the Source IDs should be globally unique within the network of communicating systems (made of several ECUs).⌋

6.4.8 Timeout detection

The previously mentioned mechanisms (CRC, Counter, Data ID, Length, Message Type, Message Result, and Source ID) enable to check the validity of received messages, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for messages, but instead if the receiver reads the

currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

6.4.9 Creation of the E2E header

6.4.9.1 E2E_PXXmProtect()

The function E2E_PXXmProtect() performs the steps as specified by the following diagrams in this section.

[PRS_E2E_01169]

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXmProtect() shall have the overall behavior as shown in Figure 6.23.⌋

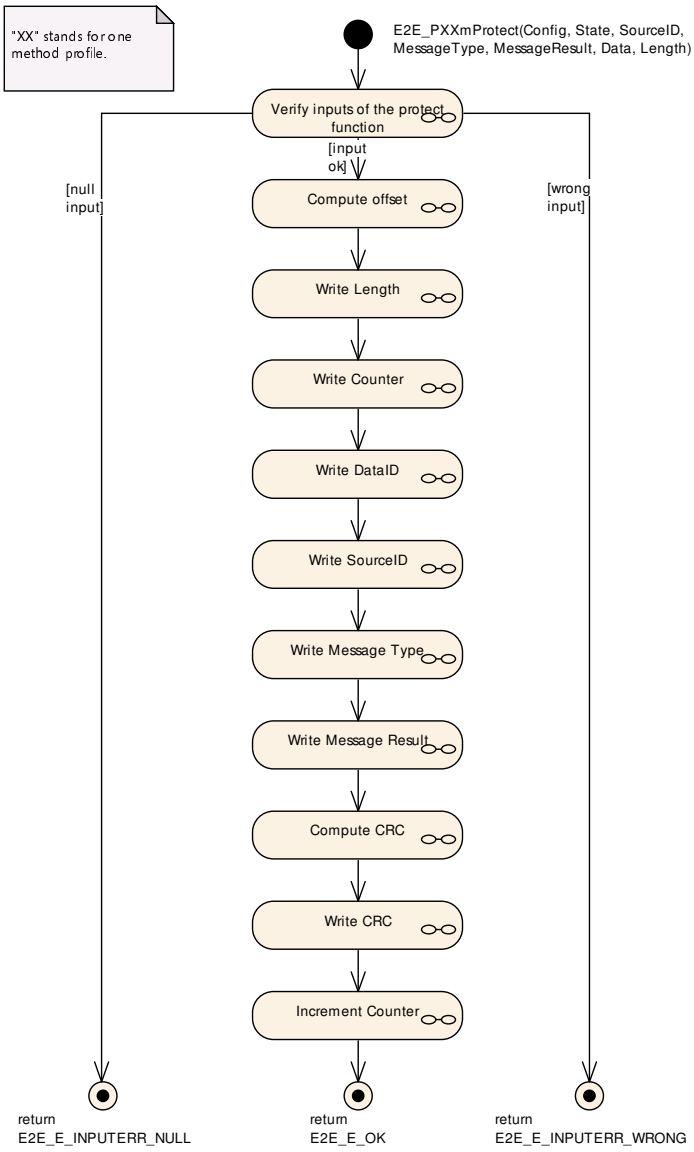

Figure 6.23: Behavior of E2E_PXXmProtect()

[PRS_E2E_01170]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_PXXmProtect() shall behave as shown in Figure 6.24.⌋

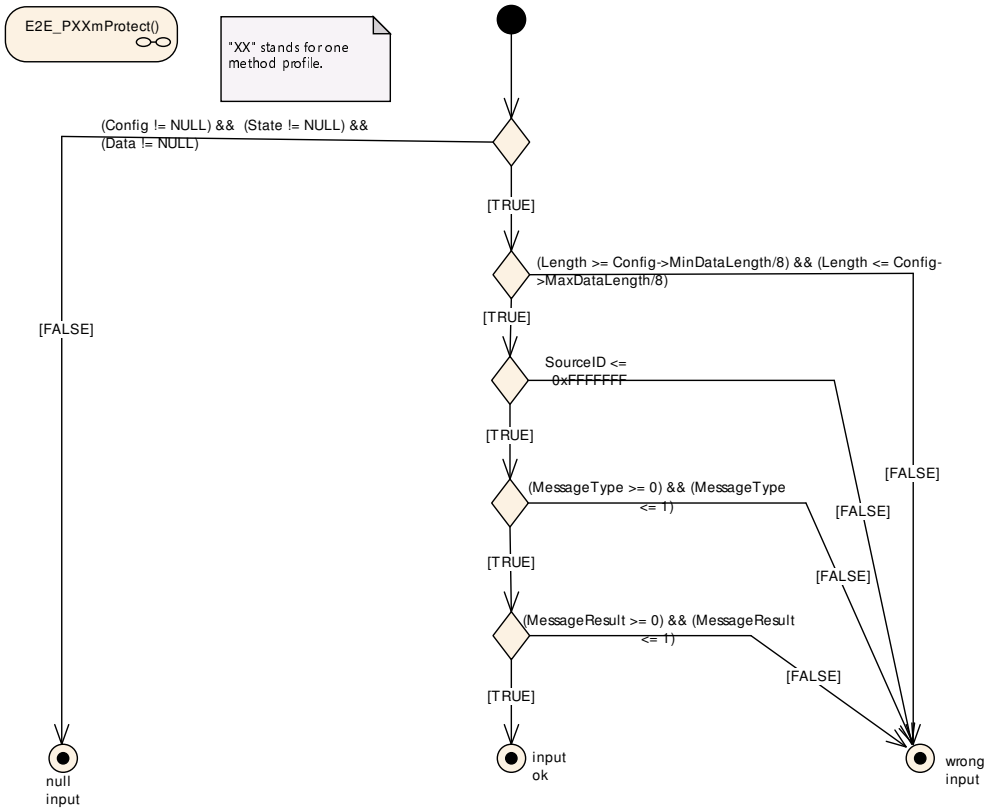

Figure 6.24: E2E_PXXmProtect() step ”Verify inputs of the protect function”

[PRS_E2E_01171]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute offset” in E2E_PXXmProtect(), E2E_PXXmForward(), E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.25.⌋

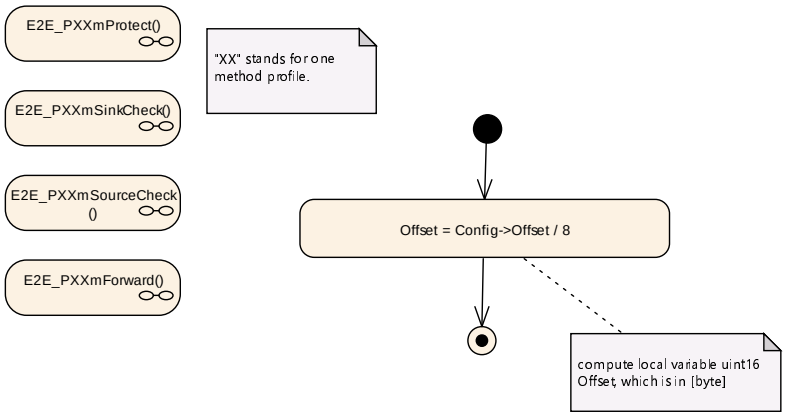

Figure 6.25: E2E_PXXmProtect(), E2E_PXXmForward(), E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() step ”Compute offset”

[PRS_E2E_01172]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Length” in E2E_PXXmProtect() and E2E_PXXmForward() shall behave as shown in Figure 6.26.⌋

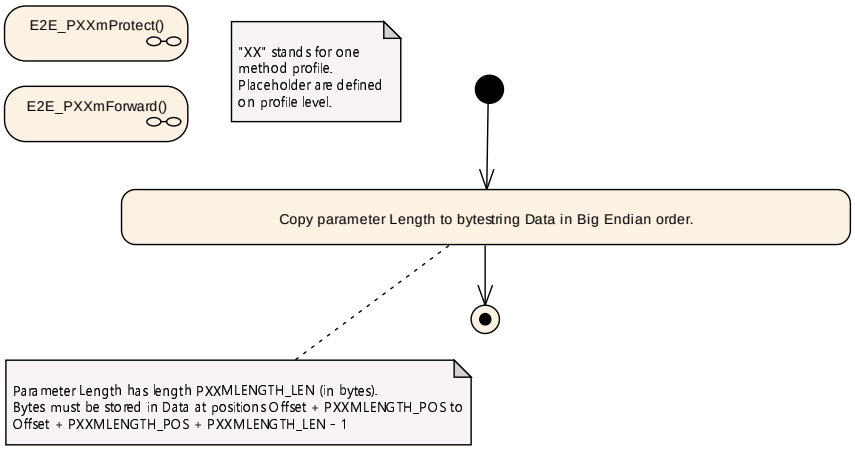

Figure 6.26: E2E_PXXmProtect() and E2E_PXXmForward() step ”Write Length”

[PRS_E2E_01173]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_PXXmProtect() shall behave as shown in Figure 6.27.⌋

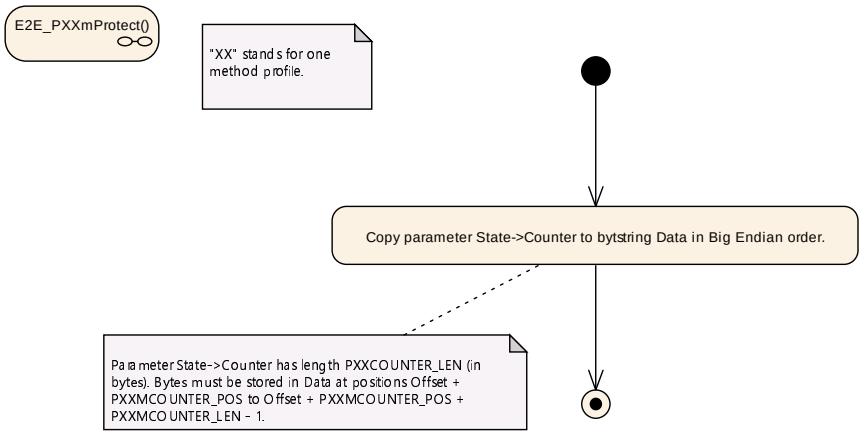

Figure 6.27: E2E_PXXmProtect() step ”Write Counter”

[PRS_E2E_01174]

Upstream requirements: RS_E2E_08539

⌈The step ”Write DataID” in E2E_PXXmProtect() shall behave as shown in Figure 6.28.⌋

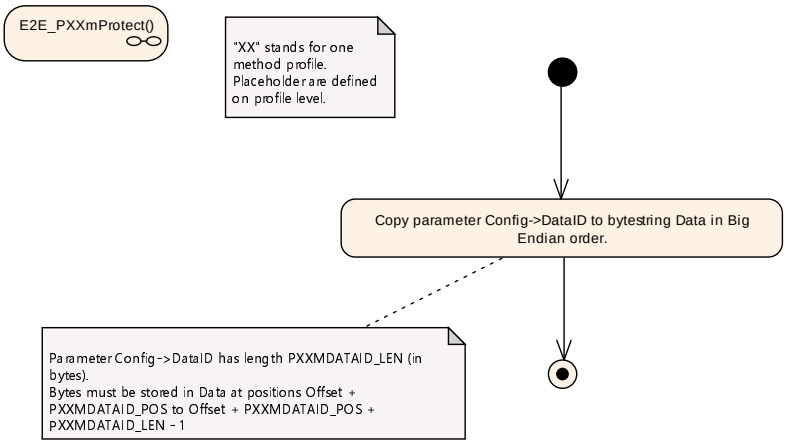

Figure 6.28: E2E_PXXmProtect() step ”Write DataID”

[PRS_E2E_01175]

Upstream requirements: RS_E2E_08539

⌈The step ”Write SourceID” in E2E_PXXmProtect() shall behave as shown in Figure 6.29.⌋

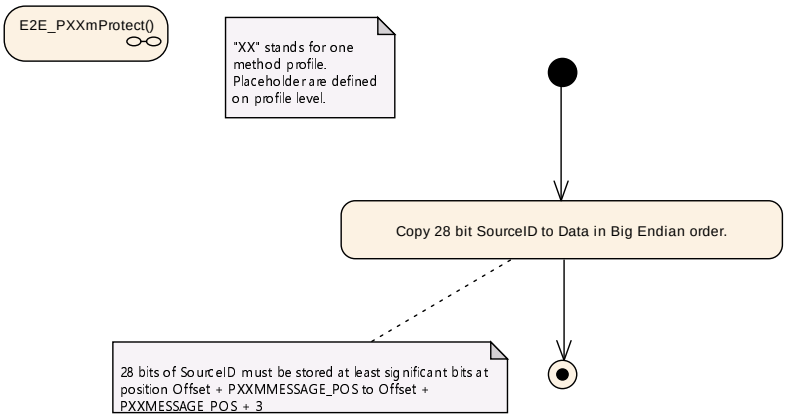

Figure 6.29: E2E_PXXmProtect() step ”Write SourceID”

[PRS_E2E_01176]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Message Type” in E2E_PXXmProtect() shall behave as shown in Figure 6.30.⌋

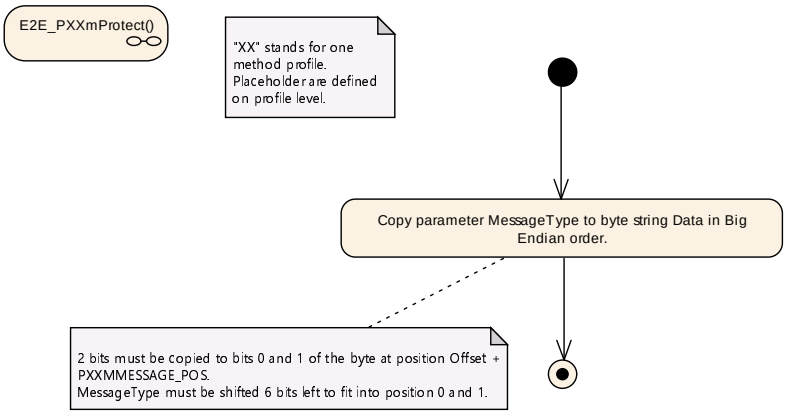

Figure 6.30: E2E_PXXmProtect() step ”Write Message Type”

[PRS_E2E_01177]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Message Result” in E2E_PXXmProtect() shall behave as shown in Figure 6.31.⌋

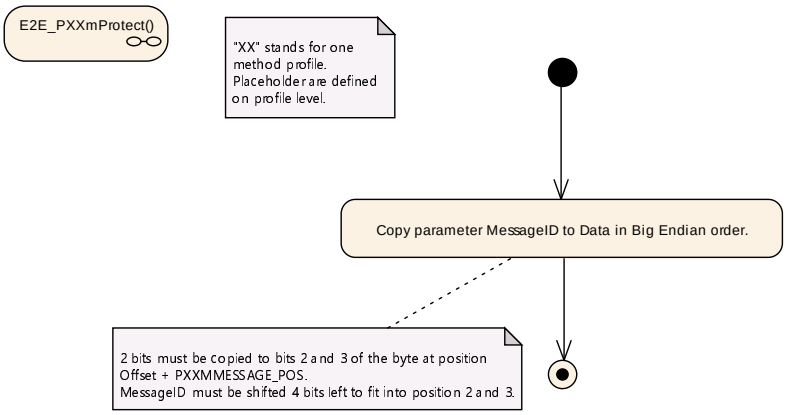

Figure 6.31: E2E_PXXmProtect() step ”Write Message Result”

[PRS_E2E_01178]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_PXXmProtect(), E2E_PXXmForward(), E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.32.⌋

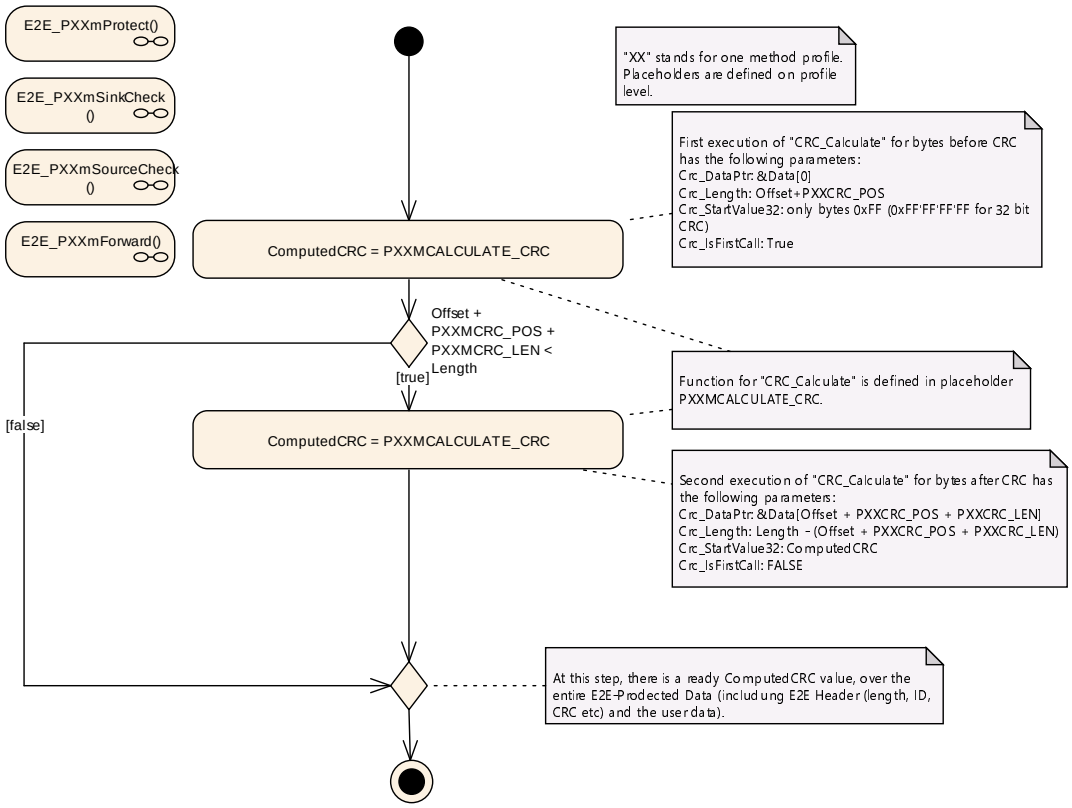

Figure 6.32: E2E_PXXmProtect(), E2E_PXXmForward(), E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() step ”Compute CRC”

[PRS_E2E_01179]

Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC”in E2E_PXXmProtect() and E2E_PXXmForward() shall behave as shown in Figure 6.33.⌋

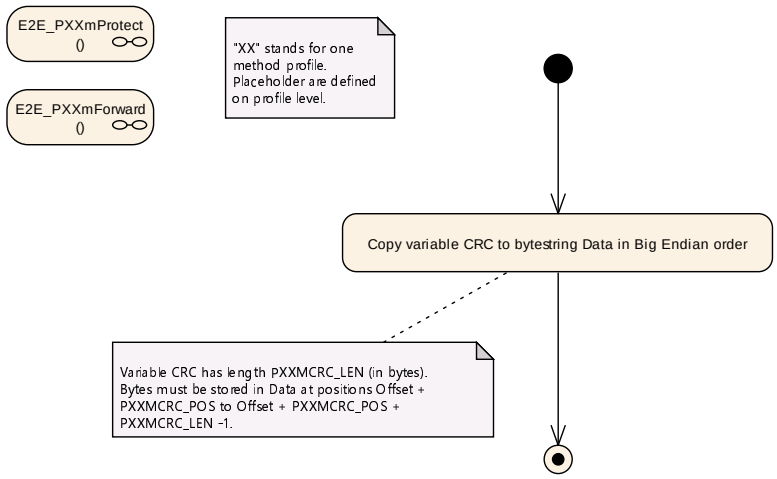

Figure 6.33: E2E_PXXmProtect() and E2E_PXXmForward() step ”Write CRC”

[PRS_E2E_01180]

Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_PXXmProtect() and E2E_PXXmForward() shall behave as shown in Figure 6.34.⌋

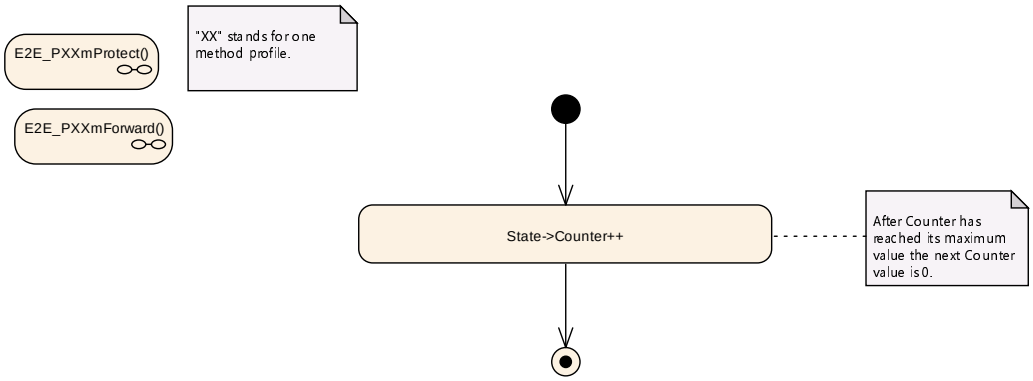

Figure 6.34: E2E_PXXmProtect() and E2E_PXXmForward() step ”Increment Counter”

6.4.9.2 E2E_PXXmForward()

The E2E_PXXmForward() function of E2E profiles for methods is called by a SW-C to protect its application data and to forward a received E2E status for use cases like translation of signal-based to service-oriented communication. If the received E2E status equals E2E_P_OK the behavior of the function shall be the same like

E2E_PXXmProtect(). The function E2E_PXXmForward() performs the steps as specified by the following diagrams in this section.

[PRS_E2E_01181] Draft

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXmForward() shall have the overall behavior as shown in Figure 6.35.⌋

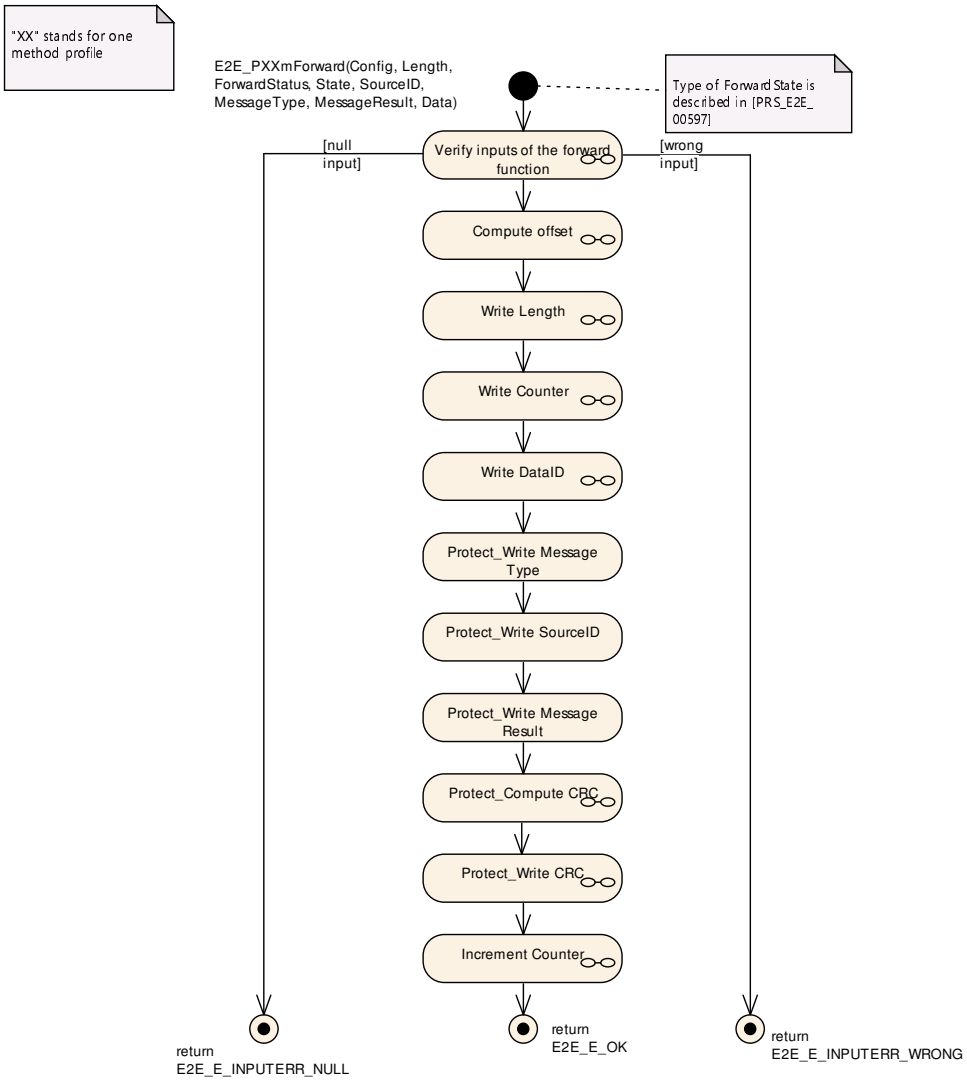

Figure 6.35: Behavior of E2E_PXXmForward()

Following steps are described in section 6.4.9.1

- ”Compute offset” see [PRS_E2E_01171]
- ”Write Length” see [PRS_E2E_01172]
- ”Write SourceID” see [PRS_E2E_01175]
- ”Write Message Type” see [PRS_E2E_01176]

- ”Write Message Result” see [PRS_E2E_01177]
- ”Compute CRC” see [PRS_E2E_01178]
- ”Write CRC” see [PRS_E2E_01179]
- ”Increment Counter” see [PRS_E2E_01180]

[PRS_E2E_01182] Draft

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the forward function” in E2E_PXXmForward() shall behave as shown in Figure 6.36.⌋

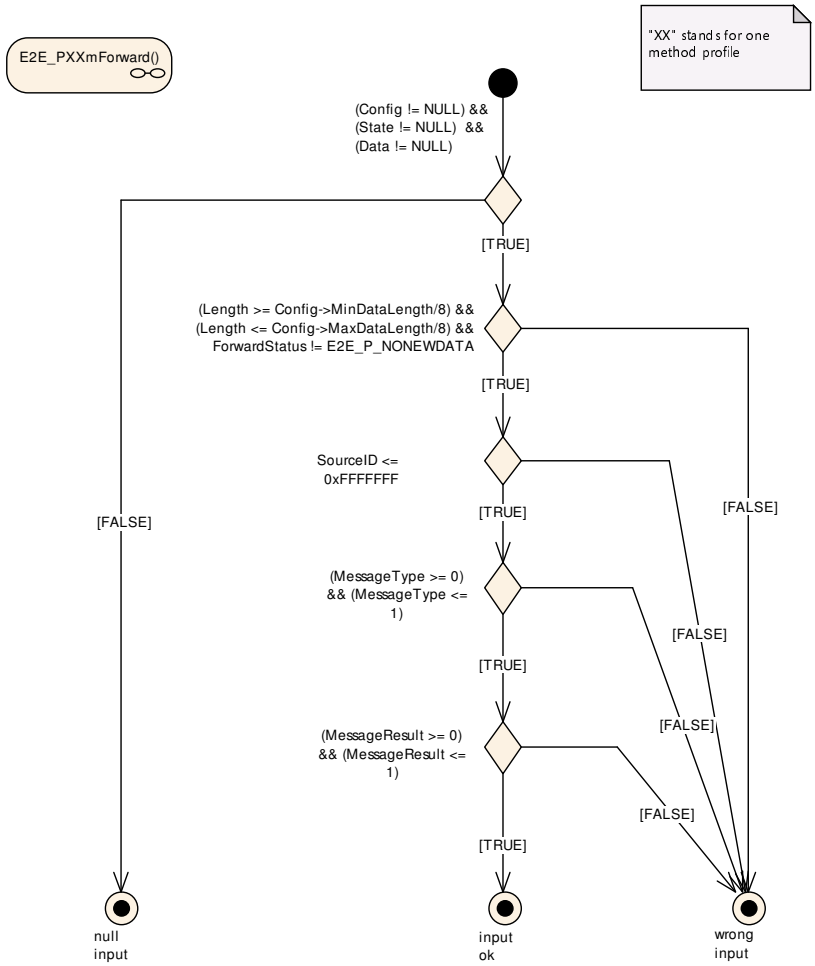

Figure 6.36: E2E_PXXmForward() step ”Verify inputs of the forward function”

[PRS_E2E_01183] Draft

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_PXXmForward() shall behave as shown in Figure 6.37.⌋

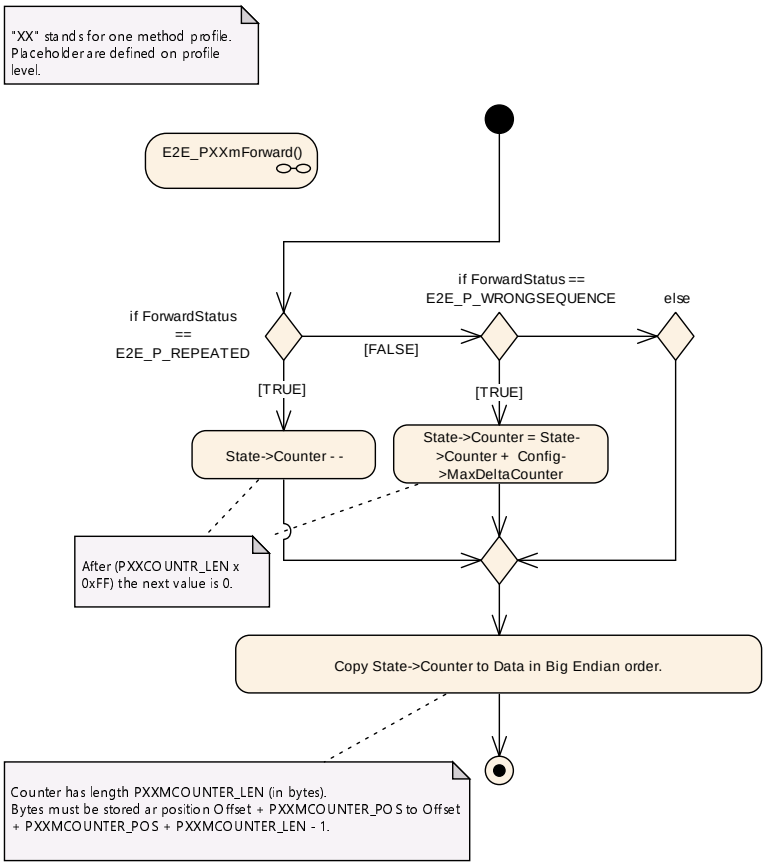

Figure 6.37: E2E_PXXmForward() step ”Write Counter”

[PRS_E2E_01184] Draft

Upstream requirements: RS_E2E_08539

⌈The step ”Write DataID” in E2E_PXXmForward() shall behave as shown in Figure 6.38.⌋

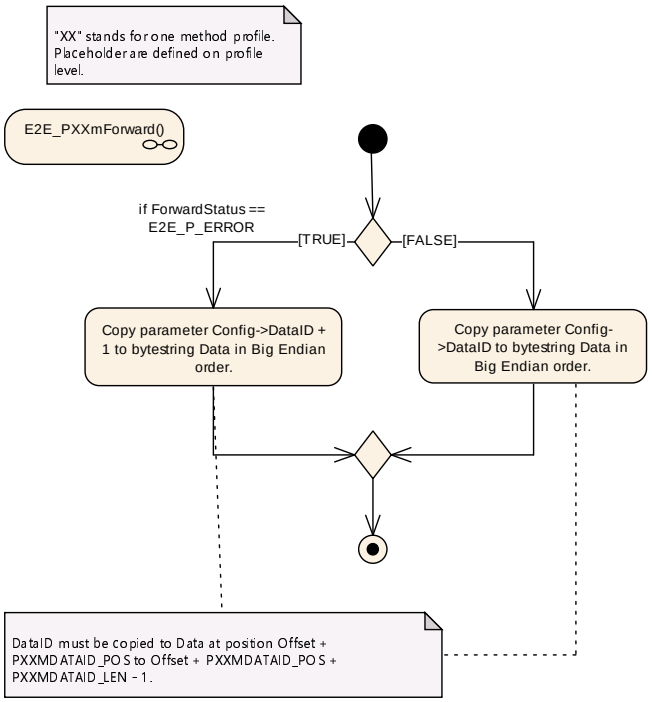

Figure 6.38: E2E_PXXmForward() step ”Write DataID”

6.4.10 Evaluation of the E2E Header

There are two check functions: E2E_PXXmSourceCheck() for the client side and E2E_PXXmSinkCheck() for the server side.

6.4.10.1 E2E_PXXmSourceCheck()

The function E2E_PXXmSourceCheck() for the client side performs the actions as specified by the following diagrams in this section.

[PRS_E2E_01185]

Upstream requirements: RS_E2E_08539

⌈"The function E2E_PXXmSourceCheck() for the client side shall have the overall behavior as shown in Figure 6.39.⌋

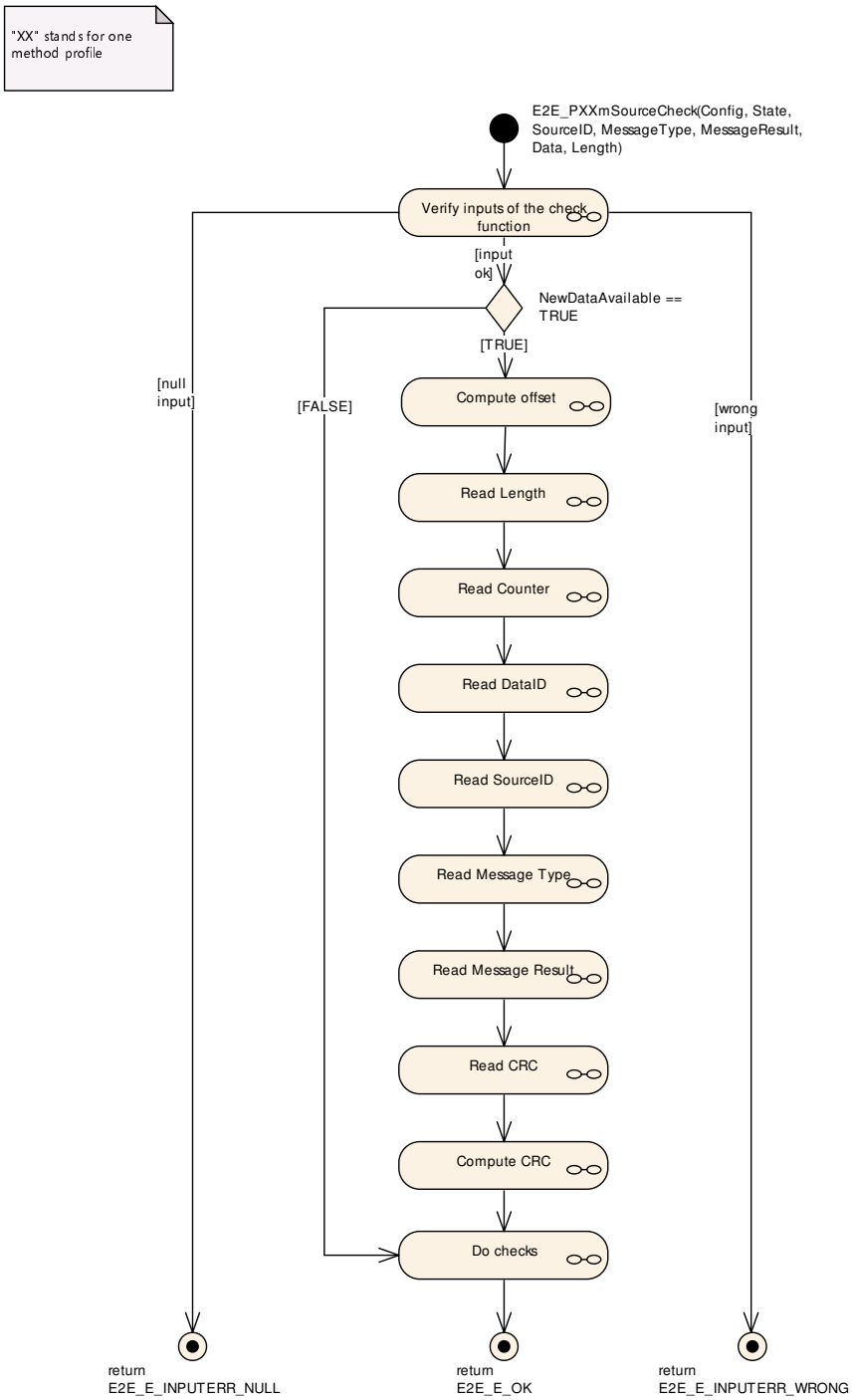

Figure 6.39: Behavior of E2E_PXXmSourceCheck()

[PRS_E2E_01186]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the function” in E2E_PXXmSourceCheck() shall behave as shown in Figure 6.40.⌋

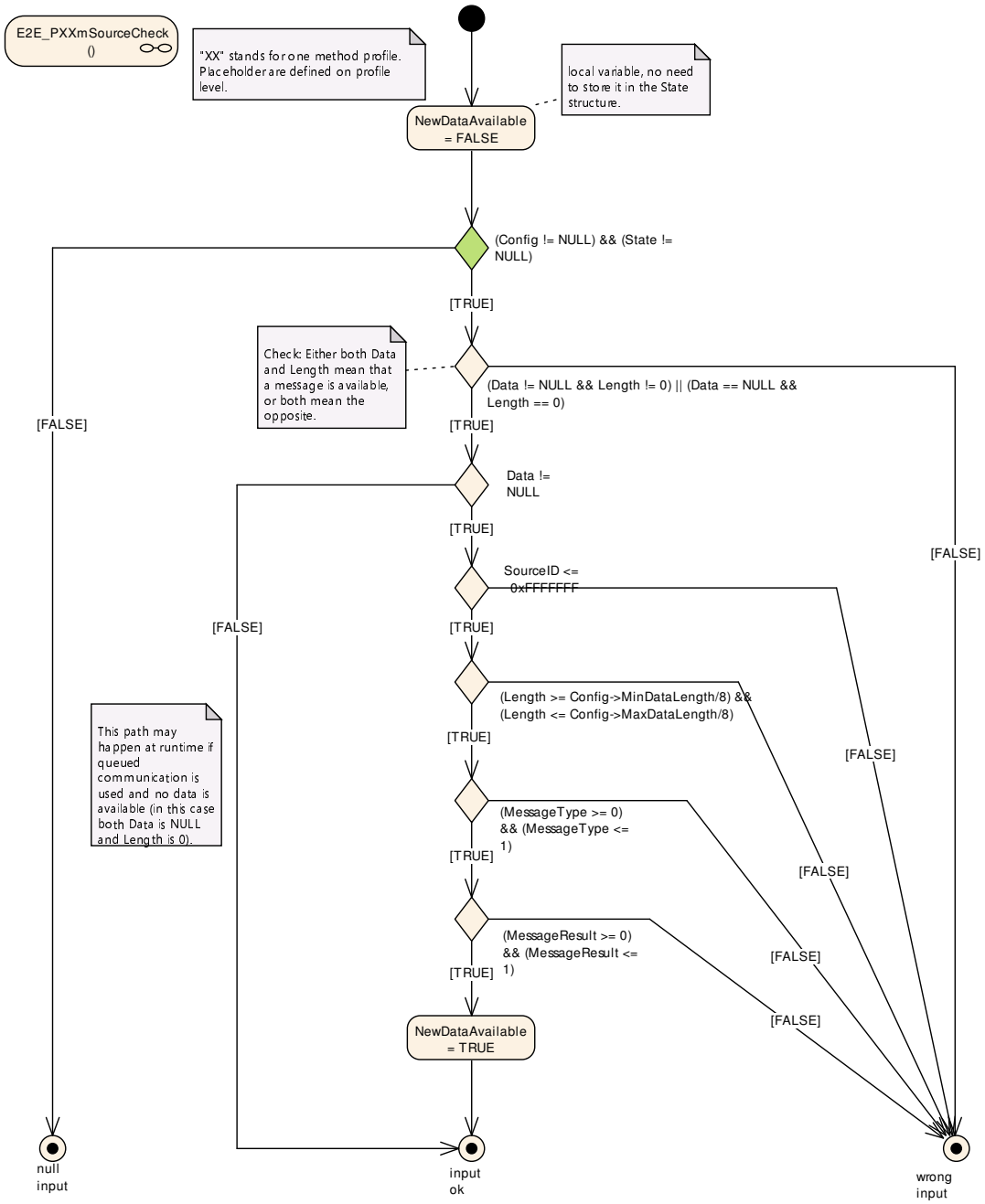

Figure 6.40: E2E_PXXmSourceCheck() step ”Verify inputs of the check function”

[PRS_E2E_01187]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Length” in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.41.⌋

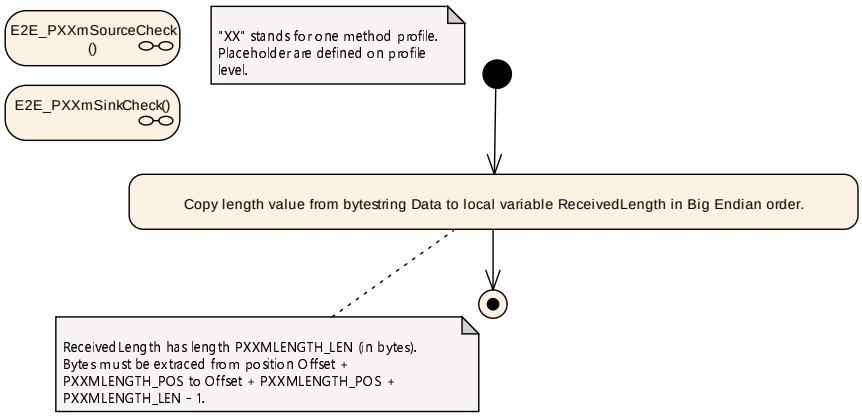

Figure 6.41: E2E_PXXmSourceCheck()/E2E_PXXmSinkCheck() step ”Read Length”

[PRS_E2E_01188]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Counter” in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.42.⌋

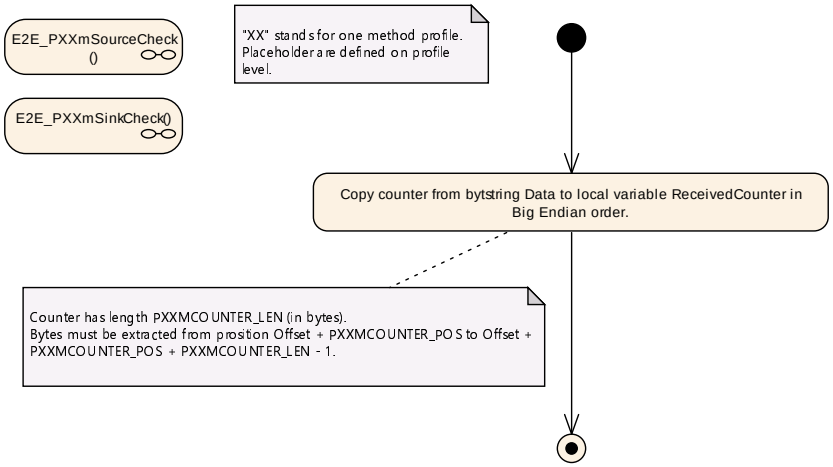

Figure 6.42: E2E_PXXmSourceCheck()/E2E_PXXmSinkCheck() step ”Read Counter”

[PRS_E2E_01189]

Upstream requirements: RS_E2E_08539

⌈The step ”Read DataID” in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.43.⌋

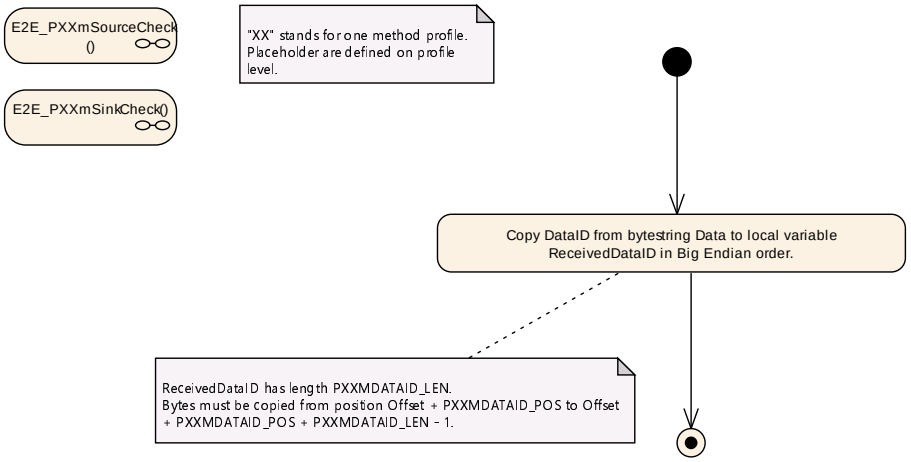

Figure 6.43: E2E_PXXmSourceCheck()/E2E_PXXmSinkCheck() step ”Read DataID”

[PRS_E2E_01190]

Upstream requirements: RS_E2E_08539

⌈The step ”Read SourceID” in E2E_PXXmSourceCheck() shall behave as shown in Figure 6.44.⌋

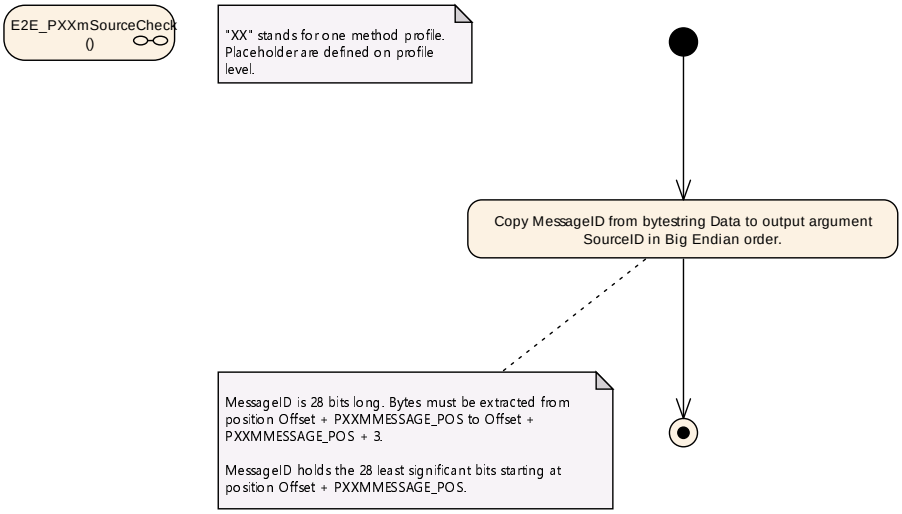

Figure 6.44: E2E_PXXmSourceCheck() step ”Read SourceID”

[PRS_E2E_01191]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Message Type” in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.45.⌋

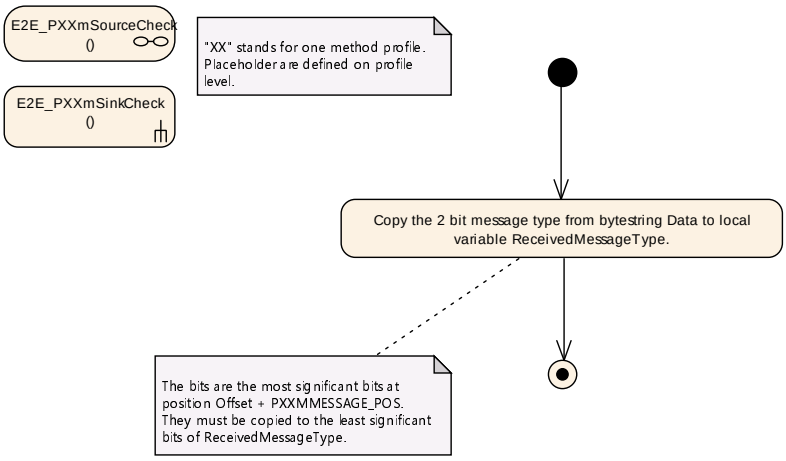

Figure 6.45: E2E_PXXmSourceCheck()/E2E_PXXmSinkCheck() step ”Read Message Type”

[PRS_E2E_01192]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Message Result” in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.46.⌋

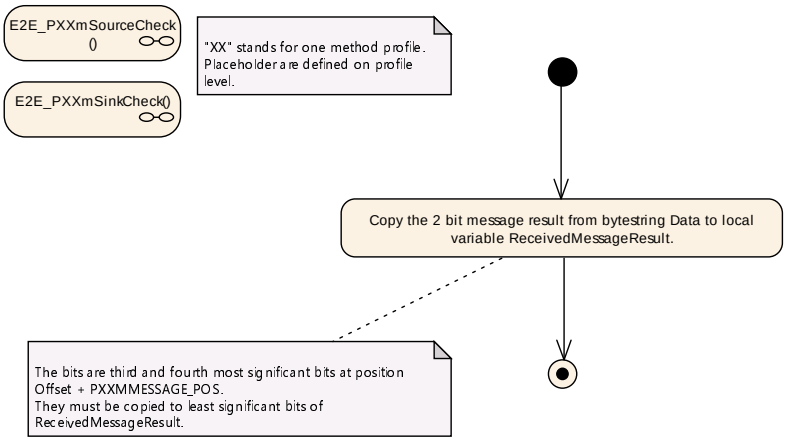

Figure 6.46: E2E_PXXmSourceCheck()/E2E_PXXmSinkCheck() step ”Read Message Result”

[PRS_E2E_01193]

Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall behave as shown in Figure 6.47.⌋

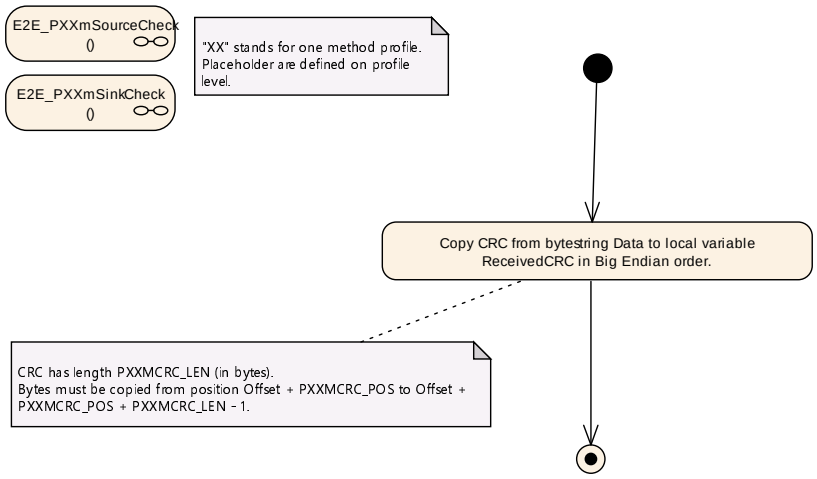

Figure 6.47: E2E_PXXmSourceCheck()/E2E_PXXmSinkCheck() step ”Read CRC”

[PRS_E2E_01194]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks” in E2E_PXXmSourceCheck() shall behave as shown in Figure 6.48.⌋

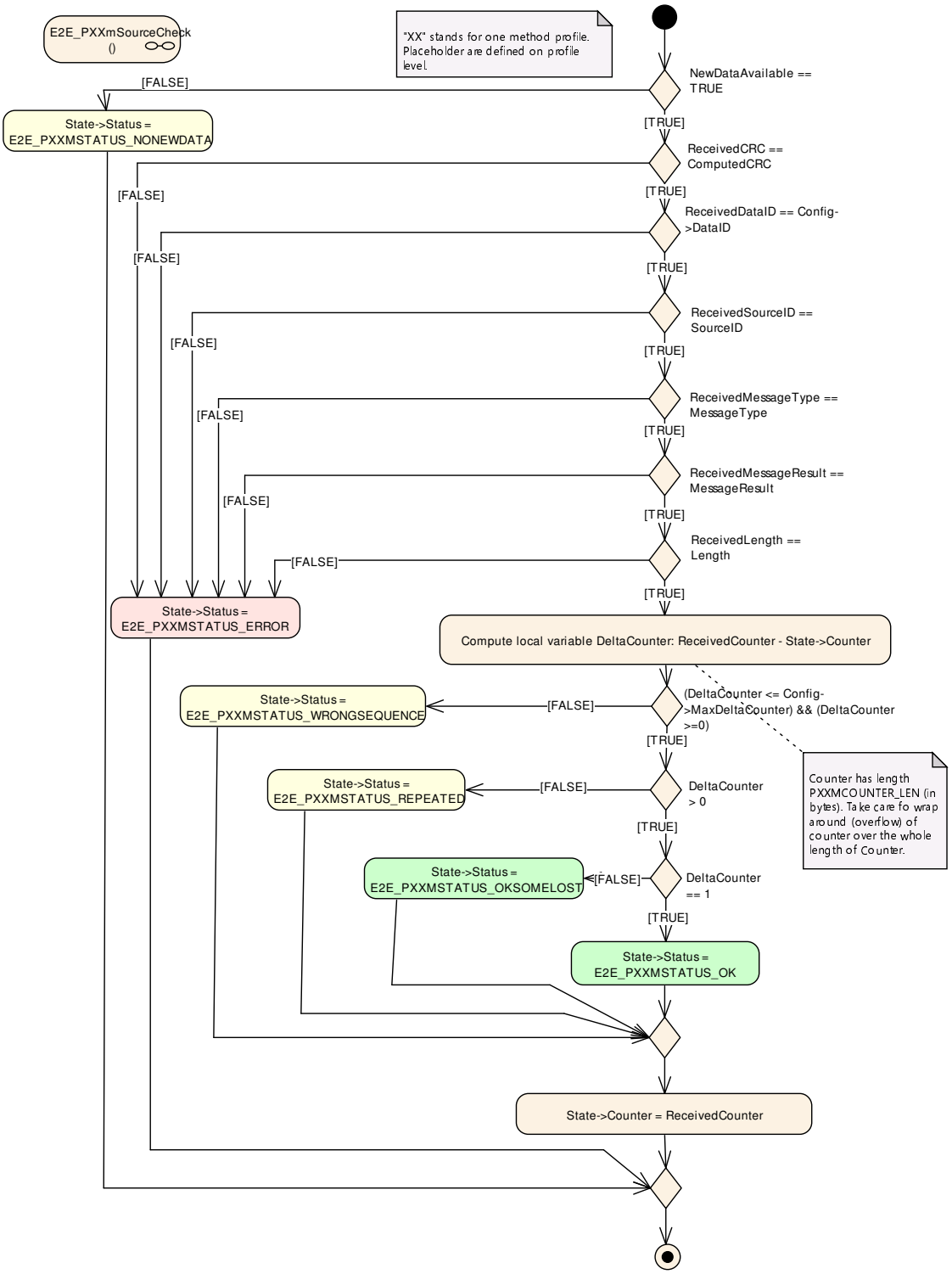

Figure 6.48: E2E_PXXmSourceCheck() step ”Do Checks”

6.4.10.2 E2E_PXXmSinkCheck()

The function E2E_PXXmSinkCheck() for the server side performs the actions as specified by the following diagrams in this section.

[PRS_E2E_01195]

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXmSinkCheck() for the server side shall have the overall behavior as shown in Figure 6.49.⌋

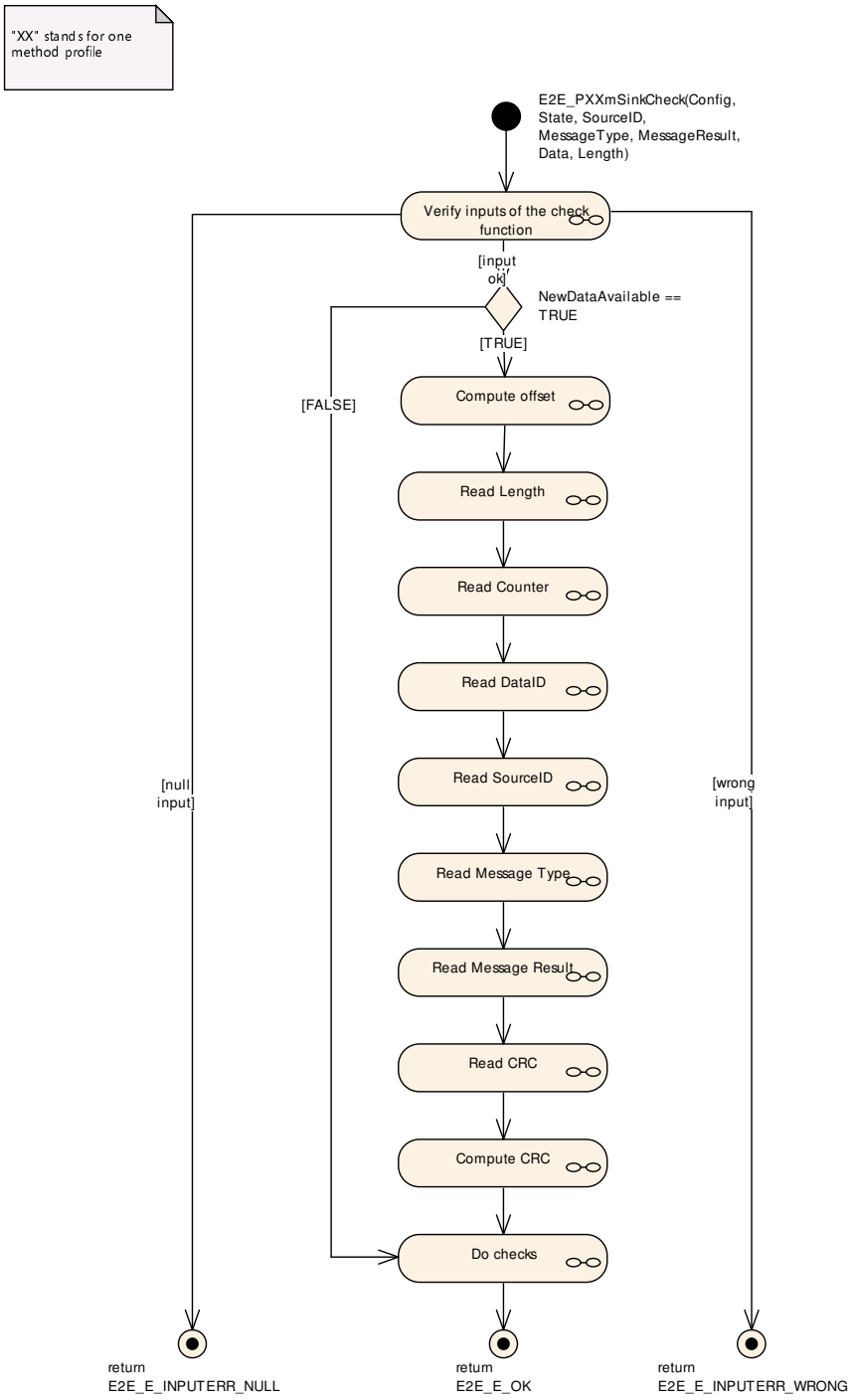

Figure 6.49: Behavior of E2E_PXXmSinkCheck()

Following steps are the same as for the client side and thus already described previously

- ”Read Length” see [PRS_E2E_01187]
- ”Read Counter” see [PRS_E2E_01188]
- ”Read DataID” see [PRS_E2E_01189]
- ”Read Message Type” see [PRS_E2E_01191]
- ”Read Message Result” see [PRS_E2E_01192]
- ”Read CRC” see [PRS_E2E_01193]
- ”Increment Counter” see [PRS_E2E_01180]

[PRS_E2E_01196]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the check function” in E2E_PXXmSinkCheck() shall behave as shown in Figure 6.50.⌋

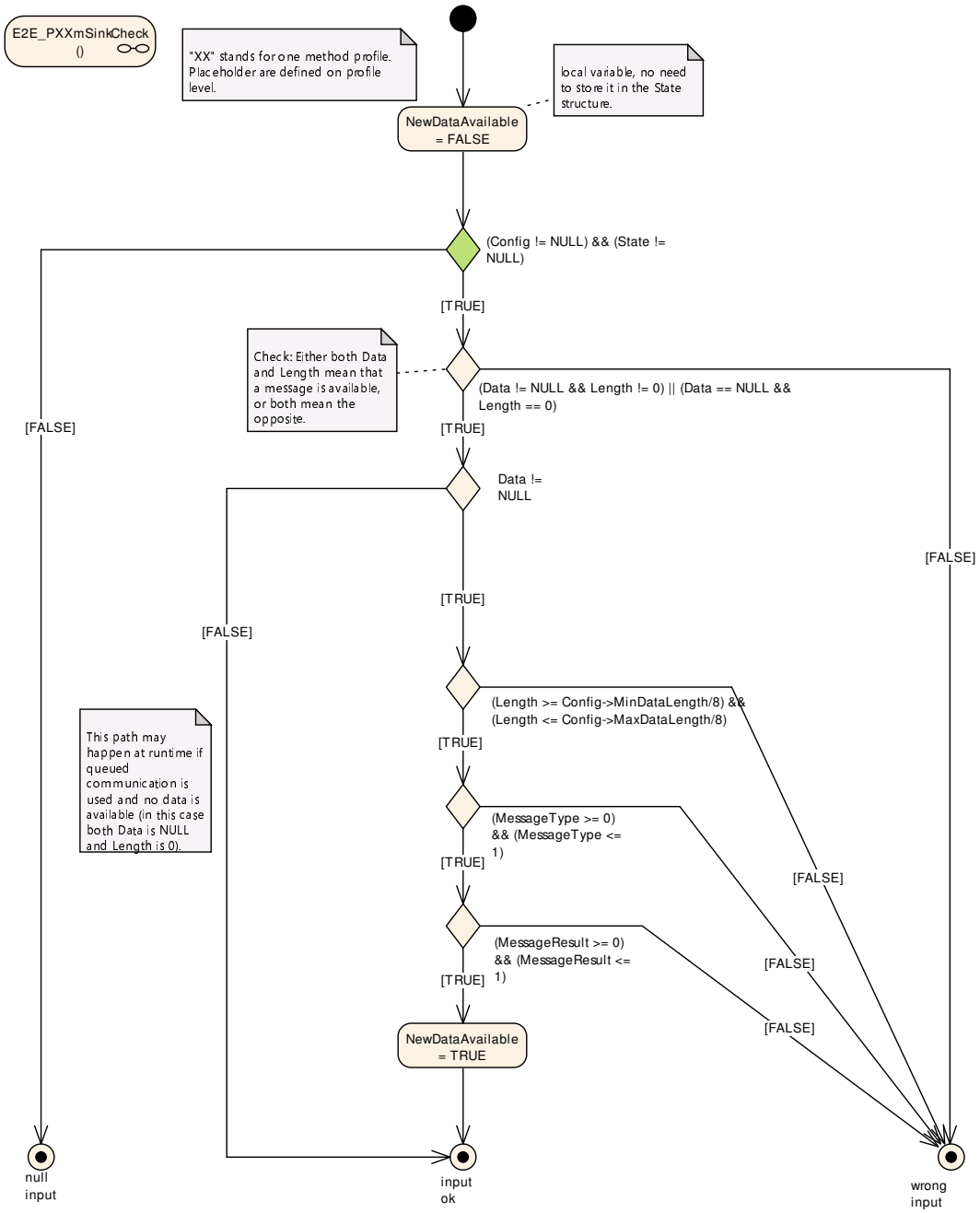

Figure 6.50: E2E_PXXmSinkCheck() step 'Verify inputs of the check function"

[PRS_E2E_01197]

Upstream requirements: RS_E2E_08539

⌈The step "Read SourceID" in E2E_PXXmSinkCheck() shall behave as shown in Figure 6.51.⌋

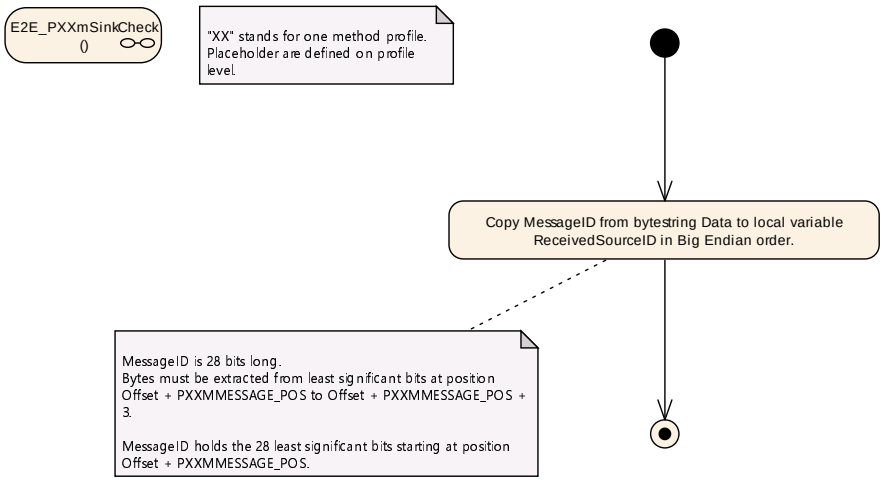

Figure 6.51: E2E_PXXmSinkCheck() step ”Read SourceID”

[PRS_E2E_01198]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks” in E2E_PXXmSinkCheck() shall behave as shown in Figure 6.52.⌋

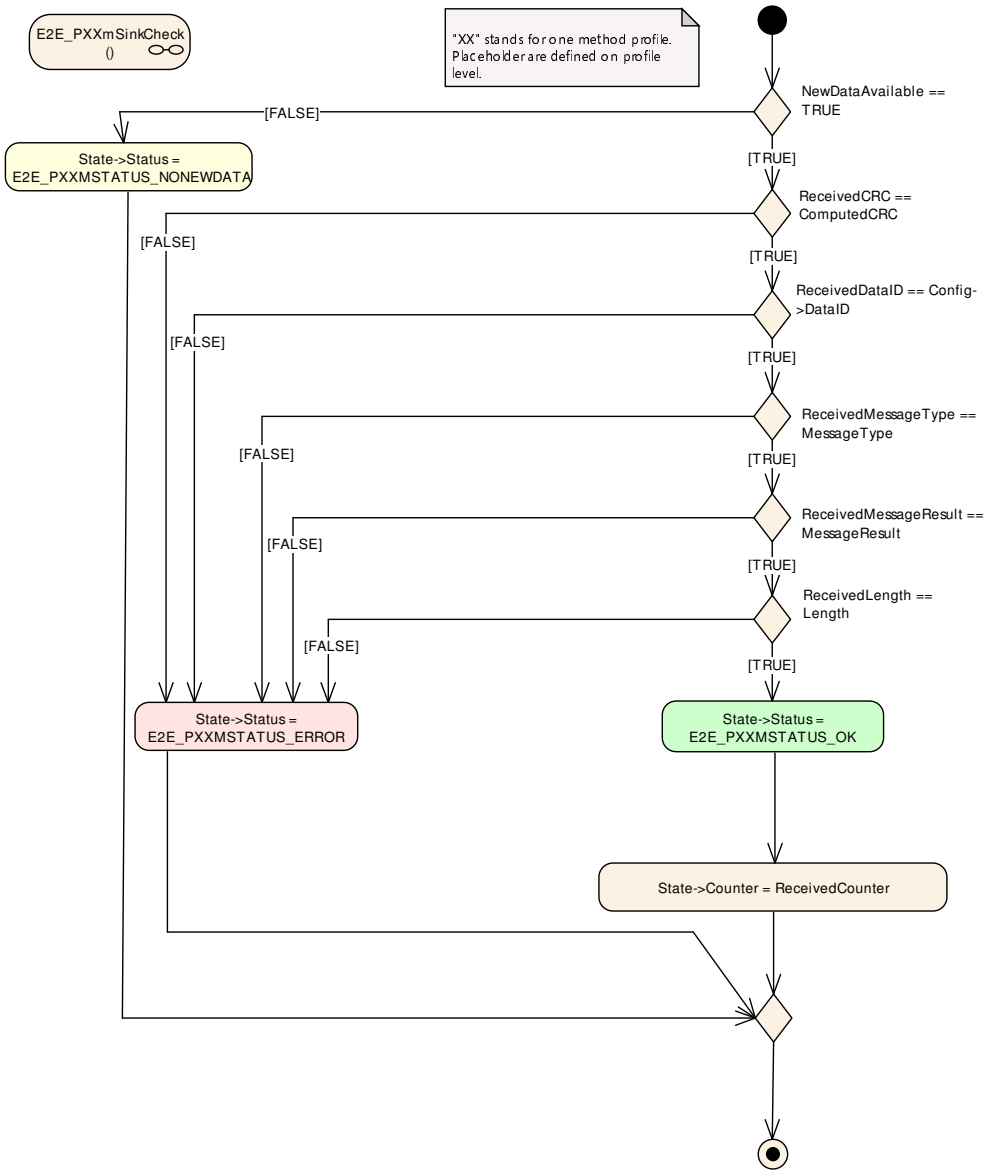

Figure 6.52: E2E_PXXmSinkCheck() step ”Do Checks”

6.4.11 Get Header Info

[PRS_E2E_01472] PXXm_GetHeaderInfo

Upstream requirements: RS_E2E_08539

⌈The function E2E_PXXmGetHeaderInfo() shall behave as shown in Figure 6.53.⌋

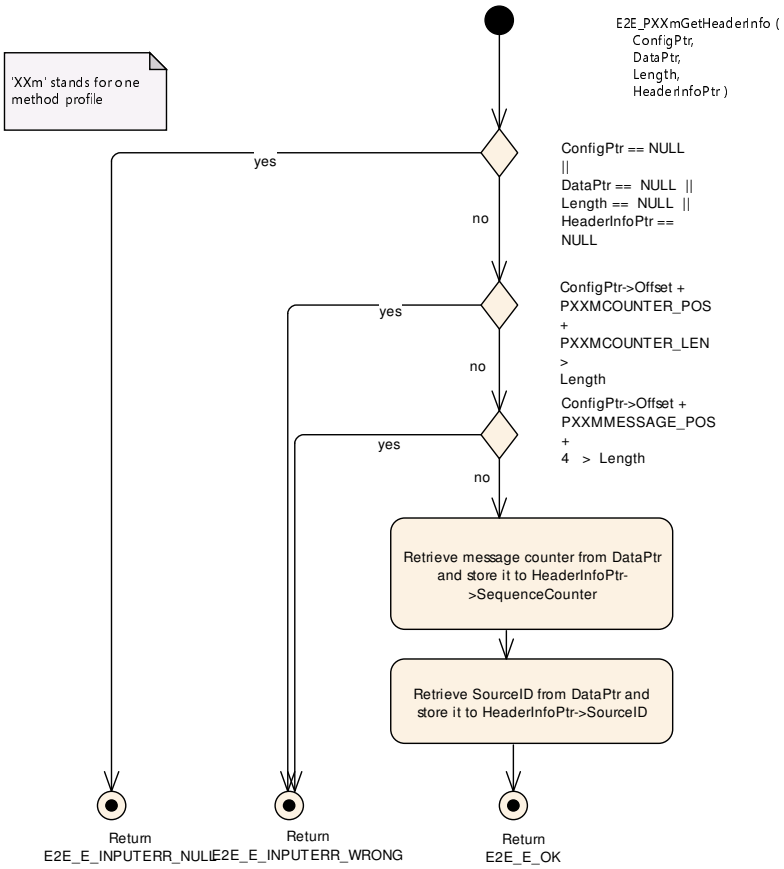

Figure 6.53: PXXm_GetHeaderInfo

6.4.12 Profile Data Types

6.4.12.1 Profile XXm Protect State Type

[PRS_E2E_01199]

Upstream requirements: RS_E2E_08528

⌈The E2E_PXXmProtect() and E2E_PXXmForward() functions' "state" shall have the members defined in [PRS_E2E_00862].⌋

[PRS_E2E_00862] E2E Profile XXm Protect State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description

Counter	Unsigned Integer	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_PXXmProtect() is called, it increments the counter up to its maximum value (0xFF'FF for a 16 bit counter, 0xFF'FF'FF'FF for a 32 bit counter, 0xFF'FF'FF'FF' FF'FF'FF'FF for a 64 bit counter). After the maximum value is reached, the next value is 0x0. The overflow is not reported to the caller.

⌋

6.4.12.2 Profile XXm Check State Type

[PRS_E2E_01200]

Upstream requirements: RS_E2E_08528

⌈The E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() functions' "state" shall have the members defined in [PRS_E2E_00863].⌋

[PRS_E2E_00863] E2E Profile XXm Check State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
Counter	Unsigned Integer	Counter of the data in previous cycle.
Status	Enumeration	Result of the verification of the Data in this cycle, determined by the Check function.

⌋

6.4.12.3 Profile XXm Check Status Enumeration

[PRS_E2E_01201]

Upstream requirements: RS_E2E_08528

⌈The step "Do Checks" in E2E_PXXmSourceCheck() and E2E_PXXmSinkCheck() shall set State->Status to one of the following enumeration values (see [PRS_E2E_00864]).⌋

[PRS_E2E_00864] E2E Profile XXm Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_PXXMSTATUS_OK	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
E2E_PXXMSTATUS_NONEWDATA	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_PXXMSTATUS_REPEATED.
E2E_PXXMSTATUS_ERROR	Error	Error not related to counters occurred (e.g. wrong CRC, wrong Length, wrong Options, wrong Data ID).
E2E_PXXMSTATUS_REPEATED	Error	The checks of the Data in this cycle were successful, except for the repetition.
E2E_PXXMSTATUS_OKSOMELOST	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented within the allowed configured delta).
E2E_PXXMSTATUS_WRONGSEQUENCE	Error	The checks of the Data in this cycle were successful, except for a counter jump, which changed more than the allowed delta

⌋

6.4.12.4 GetHeaderInfo Data Types

[PRS_E2E_01468] Data Types of PXXm_GetHeaderInfo

Upstream requirements: RS_E2E_08528

⌈The PXXm_GetHeaderInfo type shall have the members defined in [PRS_E2E_01469].⌋

[PRS_E2E_01469] Data Types Description

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
SequenceCounter	Unsigned Integer	Counter to be retrieved from Data
SpurceID	Unsigned Integer	SourceID to be retrieved from Data

⌋

6.5 Specification of E2E Profile 1

Profile 1 is a Legacy Profile and is only maintained for compatibility reasons. New Projects shall use Profile 11.

[PRS_E2E_00218]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548

⌈Profile 1 shall provide the following mechanisms: Counter, Timeout monitoring, Data ID, CRC (see [PRS_E2E_00865]).⌋

[PRS_E2E_00865] E2E Profile 1 mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548

⌈

Mechanism	Description
Counter	4bit (explicitly sent) representing numbers from 0 to 14 incremented on every send request. Both Alive Counter and Sequence Counter mechanisms are provided by E2E Profile 1, evaluating the same 4 bits.
Timeout monitoring	Timeout is determined by E2E Supervision by means of evaluation of the Counter, by a nonblocking read at the receiver. Timeout is reported by E2E Supervision to the caller by means of the status flags in E2E_P01CheckStatusType.
Data ID	16 bit, unique number, included in the CRC calculation. For dataIdMode equal to 0, 1 or 2, the Data ID is not transmitted, but included in the CRC computation (implicit transmission). For dataIdMode equal to 3: • the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits, • the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data. • the low byte is not transmitted, but it is included in the CRC computation as the first value (implicit transmission, like for dataIDMode equal to 0, 1 or 2) .

CRC	CRC-8-SAE J1850 - 0x1D (x8 + x4 + x3 + x2 + 1), but with different start and XOR values (both start value and XOR value are 0x00). This CRC is provided by CRC Supervision. Starting with AUTOSAR R4.0, the SAE8 CRC function of the CRC Supervision uses 0xFF as start value and XOR value. To compensate a different behavior of the CRC Supervision, the E2E Supervision applies additional XOR 0xFF operations starting with R4.0, to come up with 0x00 as start value and XOR value. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN.

⌋

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

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

Table 6.3: Detectable communication faults using Profile 1

6.5.1 Header Layout

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

- signals that have length < 8 bits should be allocated to one byte of an I-PDU, i.e. they should not span over two bytes.
- signals that have length >= 8 bits should start or finish at the byte limit of a message.

However, predefined E2E Profile 1 variants define specific data layouts regarding the protocol data fields, see subsection 6.3.6.

6.5.1.1 Counter

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

[1] Implementation by sender and receiver, which are using E2E-Supervision
[2] for a set of data protected by same CRC

[PRS_E2E_00075]

Upstream requirements: RS_E2E_08528

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

⌋

[PRS_E2E_00076]

Upstream requirements: RS_E2E_08528

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

⌋

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

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

6.5.1.2 Data ID

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

[PRS_E2E_00163]

Upstream requirements: RS_E2E_08528

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

1. E2E_P01_DATAID_BOTH: both two bytes (double ID configuration) are included in the CRC, first low byte and then high byte (see variant 1A - PRS_E2EProtocol_00227) or
2. E2E_P01_DATAID_ALT: depending on parity of the counter (alternating ID configuration) the high and the low byte is included (see variant 1B - PRS_E2EProtocol_00228). For even counter values the low byte is included and for odd counter values the high byte is included.

3. E2E_P01_DATAID_LOW: only the low byte is included and high byte is never used. This equals to the situation if the Data IDs (in a given application) are only 8 bits.
4. E2E_P01_DATAID_NIBBLE:
   - the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,
   - the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.
   - the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for the inclusion modes _BOTH, _ALT and _LOW)

⌋

[PRS_E2E_00085]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00169]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_LOW, the high byte of Data ID shall be set to 0x00.⌋

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

[PRS_E2E_00306]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_NIBBLE, the high nibble of the high byte shall be 0x0.⌋

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

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

6.5.1.3 CRC calculation

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

[PRS_E2E_00070]

Upstream requirements: RS_E2E_08529, RS_E2E_08533

⌈E2E Profile 1 shall use the polynomial of CRC-8-SAE J1850, i.e. the polynomial 0x1D (x8 + x4 + x3 + x2 + 1), but with start value and XOR value equal to 0x00.⌋

Note: To calculate a CRC with CRCLib function Crc_CalculateCRC8() with start value = 0x00 this function must be called with third parameter (Crc_StartValue8) equal to the complement of the intended start value. For start value = 0x00 this parameter must be equal to 0xFF.

See also [PRS_E2E_00865](E2E Profile 1 mechanisms) and Figure 6.58, used by E2E_P01Protect() and E2E_P01Check()).

For details of CRC calculation, the usage of start values and XOR values see also SWS_CRCLibrary[3].

[PRS_E2E_00190]

Upstream requirements: RS_E2E_08528

⌈E2E Profile 1 shall use the Crc_CalculateCRC8 () function of the SWS CRC Supervision for calculating CRC checksums.⌋

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

The CRC calculation is illustrated by the following two examples.

Figure 6.54 and Figure 6.55 show how the CRC for signal based communication would be calculated. Figure 6.54 uses the following configuration

1. CRC is the 0th byte in message (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_BOTH
4. unusedBitPattern = 0xFF.

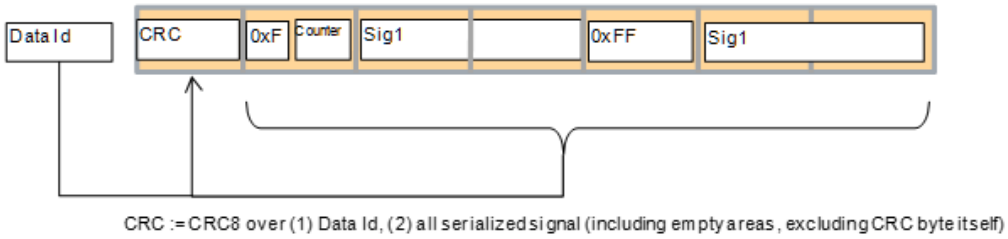

Figure 6.54: E2E Profile 1 variant 1A CRC calculation example

Figure 6.55 uses the following configuration

1. CRC is the 0th byte in the message (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1st byte (i.e. starts with bit offset 12)
4. E2E_P01DataIDMode = E2E_P01_DATAID_NIBBLE
5. unusedBitPattern = 0xFF.

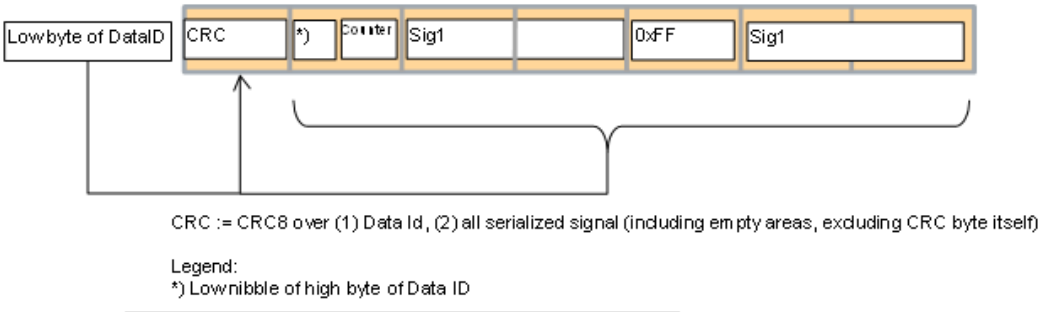

Figure 6.55: E2E Profile 1 variant 1C CRC calculation example

The Data ID can be encoded in CRC in different ways, see [PRS_E2E_00163].

[PRS_E2E_00082]

Upstream requirements: RS_E2E_08531

⌈In E2E Profile 1, the CRC is calculated over:

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

⌋

[PRS_E2E_00640]

Upstream requirements: RS_E2E_08539

⌈If DataIDMode is set to E2E_P01_DATAID_NIBBLE, the CRC calculation shall be done by first calculating over the low byte of the Data ID, then a zero-byte (0x00), and then the user data.⌋

6.5.2 Creation of E2E-Header

6.5.2.1 E2E_P01Protect

[PRS_E2E_00195]

Upstream requirements: RS_E2E_08528

⌈The function E2E_P01Protect() shall:

1. write the Counter in Data,
2. write DataID nibble in Data, if E2E_P01_DATAID_NIBBLE configuration is used
3. compute the CRC
4. write CRC in Data
5. increment the Counter (which will be used in the next invocation of E2E_P01Protect()),as specified by Figure 6.56 and Figure 6.57

⌋

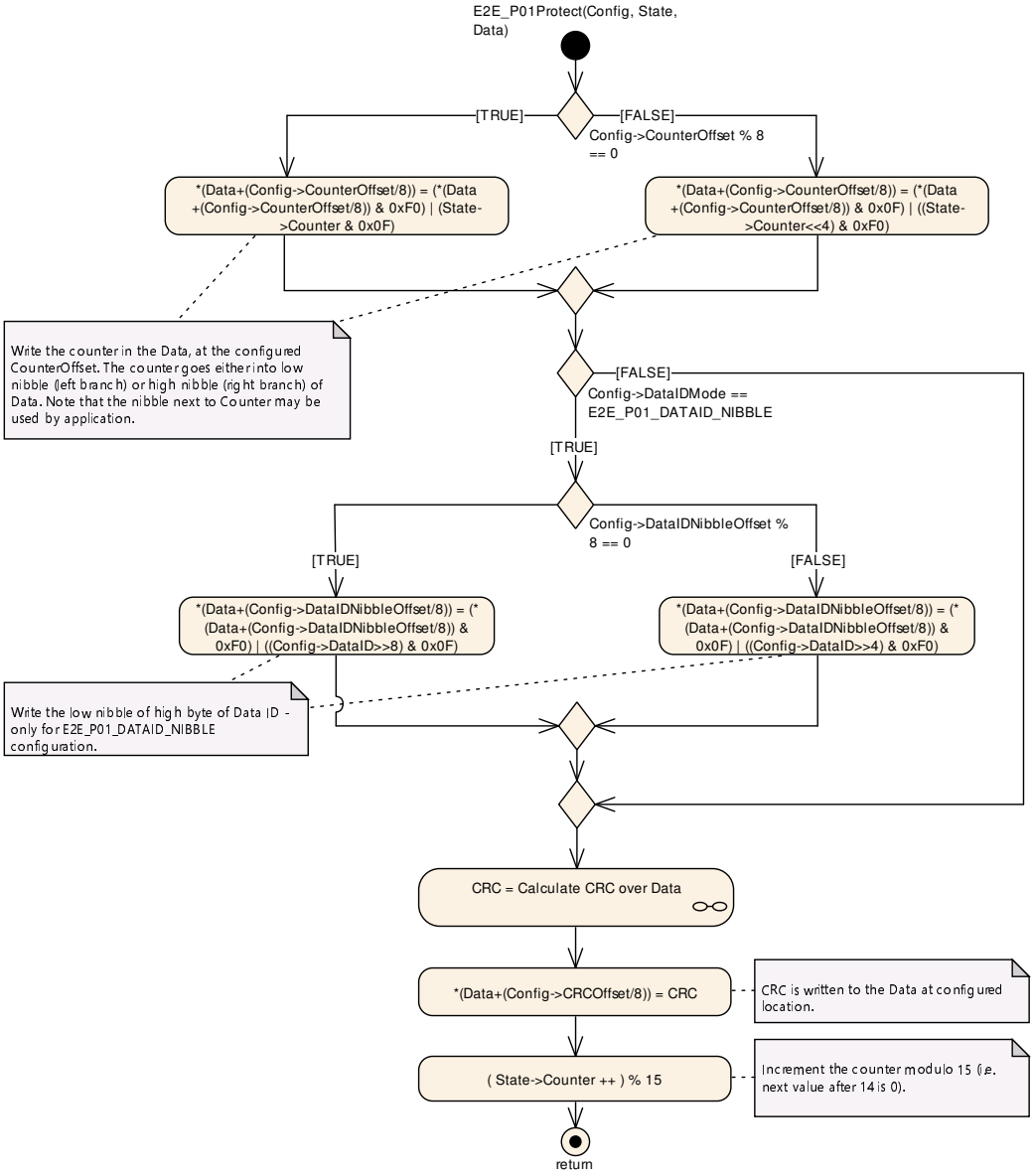

Figure 6.56: E2E_P01Protect()

6.5.2.2 Calculate CRC

The diagram of the function E2E_P01Protect() (see above chapter), E2E_P01Forward() and E2E_P01Check() (see below chapters) have a sub-diagram specifying the calculation of CRC:

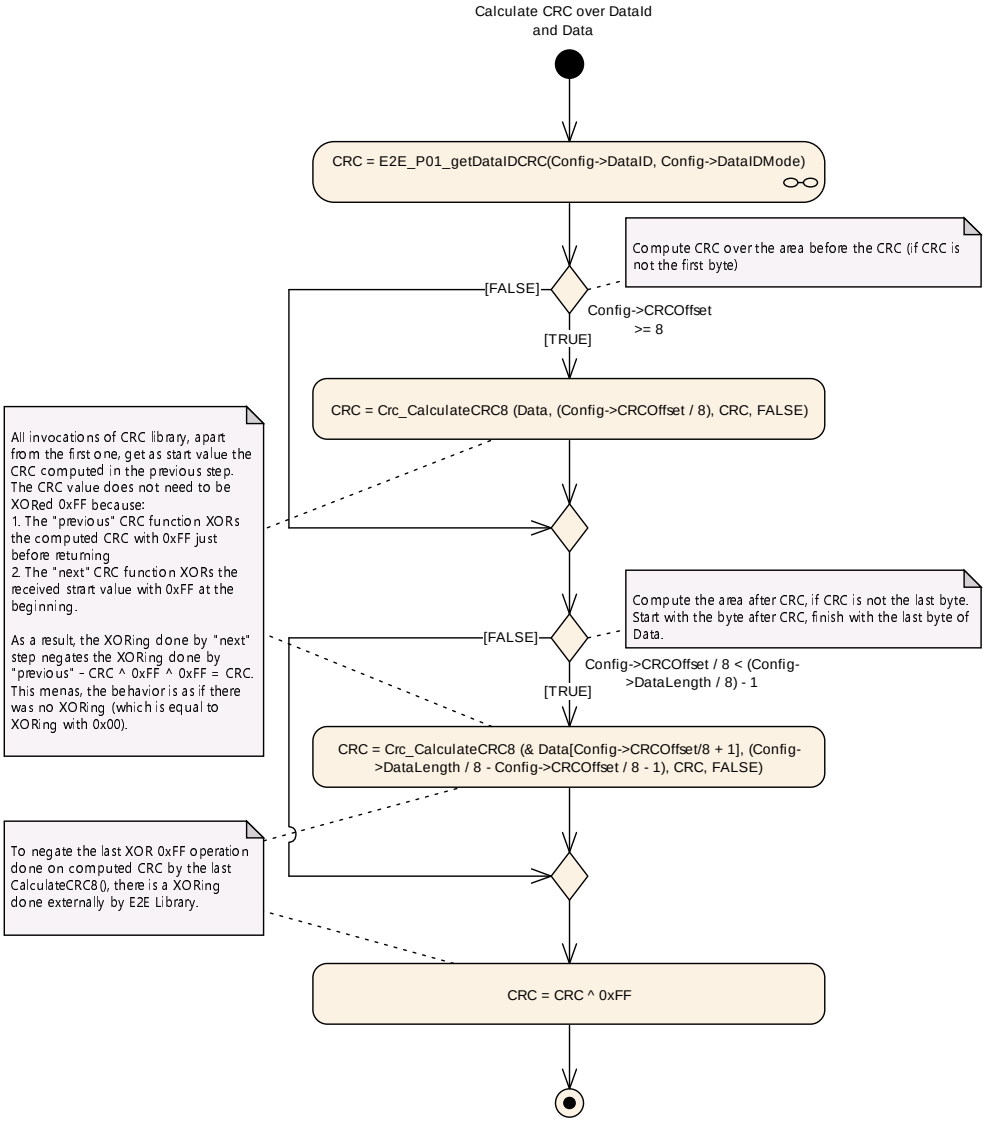

Figure 6.57: Subdiagram „Calculate CRC over Data ID and Data", used by E2E_P01Protect(), E2E_P01Forward() and E2E_P01Check()

The diagram of the function "Calculate CRC over Data ID and Data" has a sub-diagram specifying the calculation of DataID CRC, which is shown by Figure 6.58.

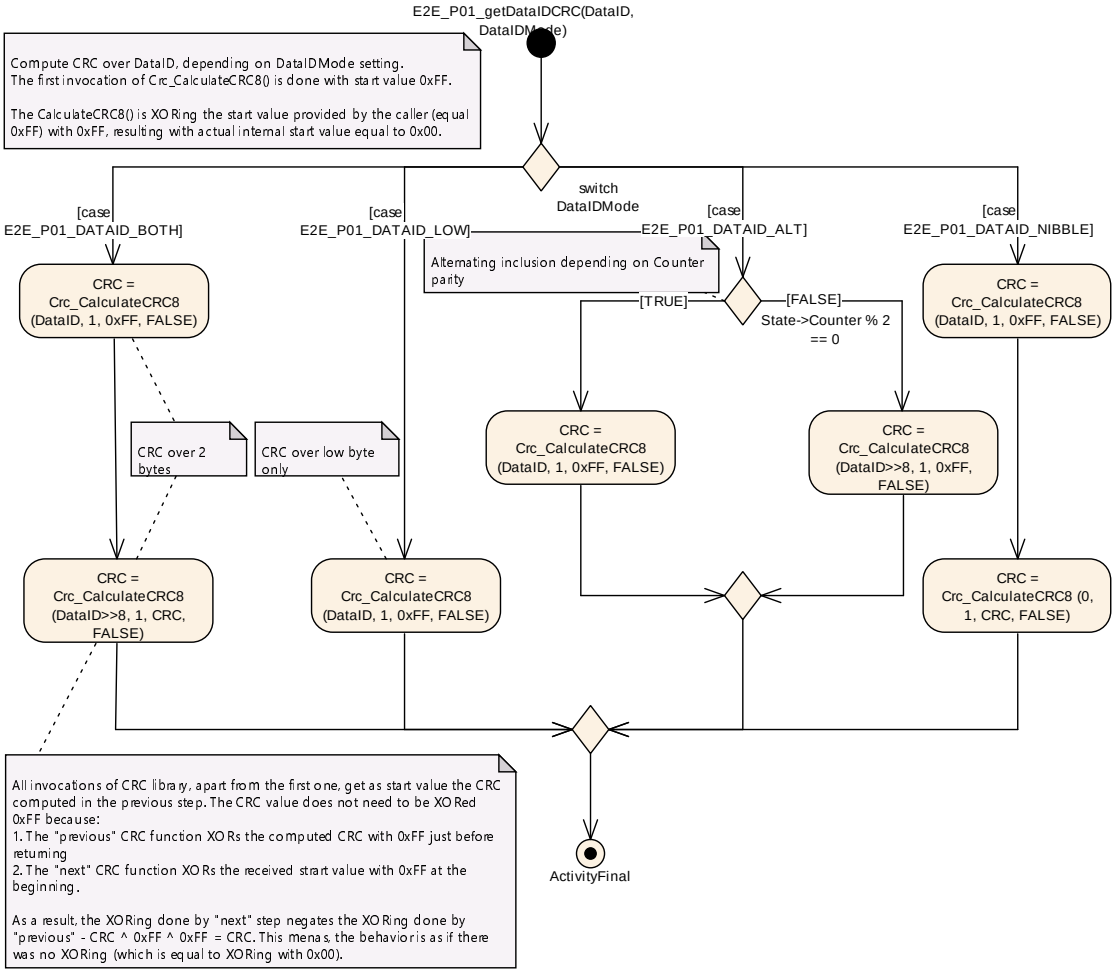

Figure 6.58: Subdiagram ”getDataIDCRC”, used by E2E_P01Protect() and E2E_P01Check()

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

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

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

To have the same results of the functions E2E_P01Protect() and E2E_P0-2-1Check() in >=R4.0 and R3.2, while using differently functioning CRC Supervision, E2E „compensates” different behavior of the CRC Supervision. This results with different invocation of the CRC Supervision by E2E Supervision Figure 6.57 in >=R4.0 and R3.2. This means Figure 6.57 is different in >=R4.0 and R3.2.

6.5.2.3 E2E_P01Forward

[PRS_E2E_00608]

Upstream requirements: RS_E2E_08528

⌈The function E2E_P01Forward() shall calculate the e2e header data based on the current value of the IN parameter ForwardStatus.⌋

The E2E_P01Forward() has additional requirements to the E2E_P01Protect() since it shall be used to reconstruct an E2E-State on an outgoing message.

[PRS_E2E_00609]

Upstream requirements: RS_E2E_08528

⌈If ForwardStatus equals to E2E_P_OK the function E2E_P01Forward() shall:

1. write the Counter in Data
2. write DataID nibble in Data, if E2E_P01_DATAID_NIBBLE configuration is used
3. compute the CRC over DataID and Data
4. write CRC in Data
5. increment the Counter (which will be used in the next invocation of E2E_P01Forward()), as specified by Figure 6.59 and Figure 6.57

⌋

[PRS_E2E_00610]

Upstream requirements: RS_E2E_08528

⌈If ForwardStatus equals to E2E_P_REPEATED the function E2E_P01Forward() shall :

1. decrement the Counter
2. write Counter in Data
3. write DataID nibble in Data, if E2E_P01_DATAID_NIBBLE configuration is used
4. compute the CRC over DataID and Data
5. write CRC in Data
6. increment the Counter (which will be used in the next invocation of E2E_P01Forward()), as specified by Figure 6.59 and Figure 6.57

⌋

[PRS_E2E_00611]

Upstream requirements: RS_E2E_08528

⌈If ForwardStatus equals to E2E_P_WRONGSEQUENCE the function E2E_P01Forward() shall use counter + MaxDeltaCounterInit :

1. calculate Counter = Counter + MaxDeltaCounterInit
2. write the Counter in Data
3. write DataID nibble in Data, if E2E_P01_DATAID_NIBBLE configuration is used
4. compute the CRC over DataID and Data
5. write CRC in Data
6. increment the Counter (which will be used in the next invocation of E2E_P01Forward()), as specified by Figure 6.59 and Figure 6.57

⌋

[PRS_E2E_00612]

Upstream requirements: RS_E2E_08528

⌈If ForwardStatus equals to E2E_P_ERROR the function E2E_P01Forward() shall use DataID + 1:

1. DataID = DataID+1
2. write the Counter in Data
3. write DataID nibble in Data, if E2E_P01_DATAID_NIBBLE configuration is used
4. compute the CRC over DataID and Data
5. write CRC in Data
6. increment the Counter (which will be used in the next invocation of E2E_P01Forward()), as specified by Figure 6.59 and Figure 6.57

⌋

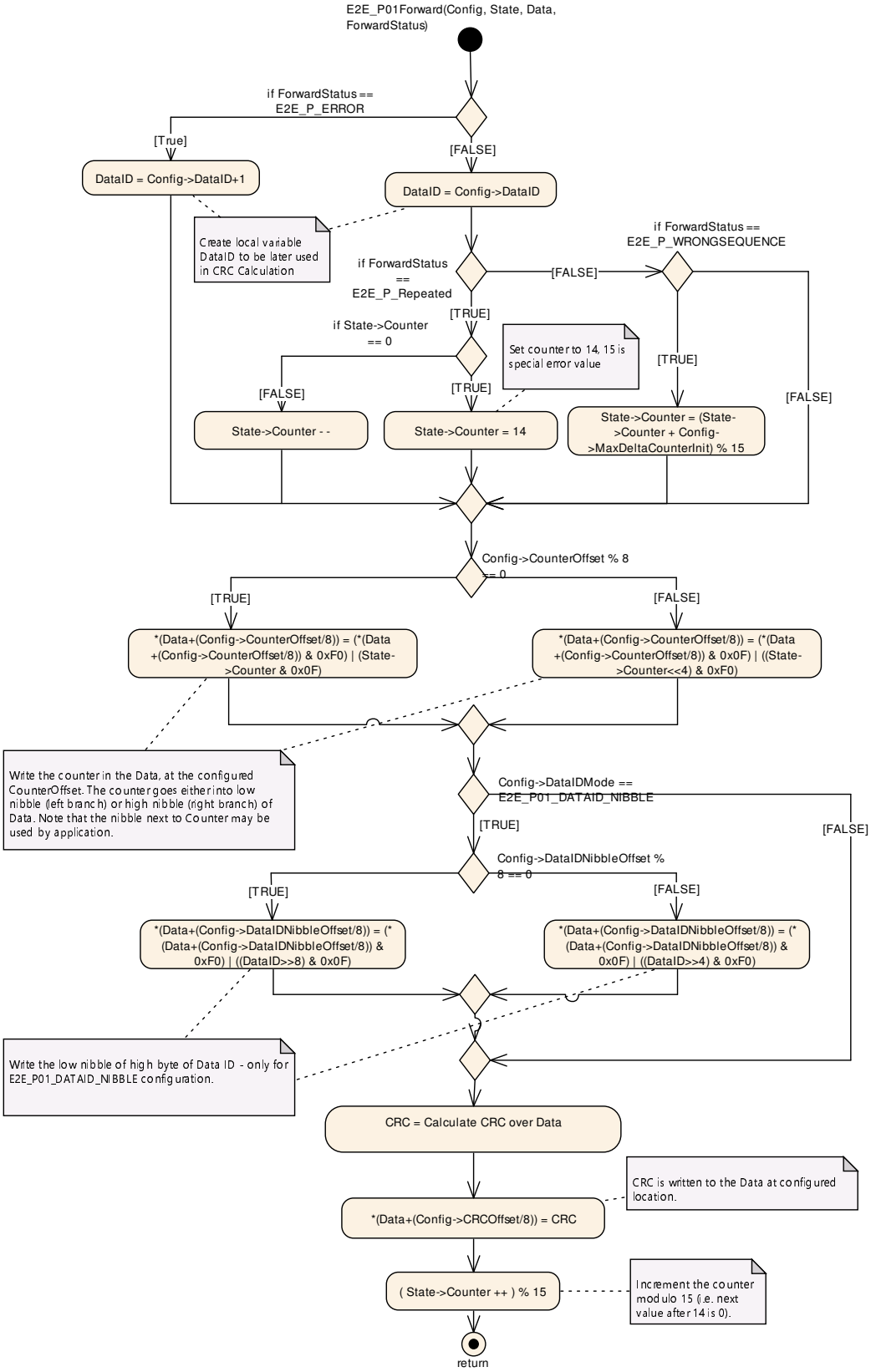

Figure 6.59: E2E_P01Forward()

6.5.3 Evaluation of E2E- Header

6.5.3.1 E2E_P01Check

[PRS_E2E_00196]

Upstream requirements: RS_E2E_08528, RS_E2E_08530

⌈The function E2E_P01Check shall

1. Check the CRC
2. Check the Data ID nibble, i.e. compare the expected value with the received value (for E2E_P01_DATAID_NIBBLE configuration only)
3. Check the Counter,
4. determine the check Status,as specified by Figure 6.60 and Figure 6.57.

⌋

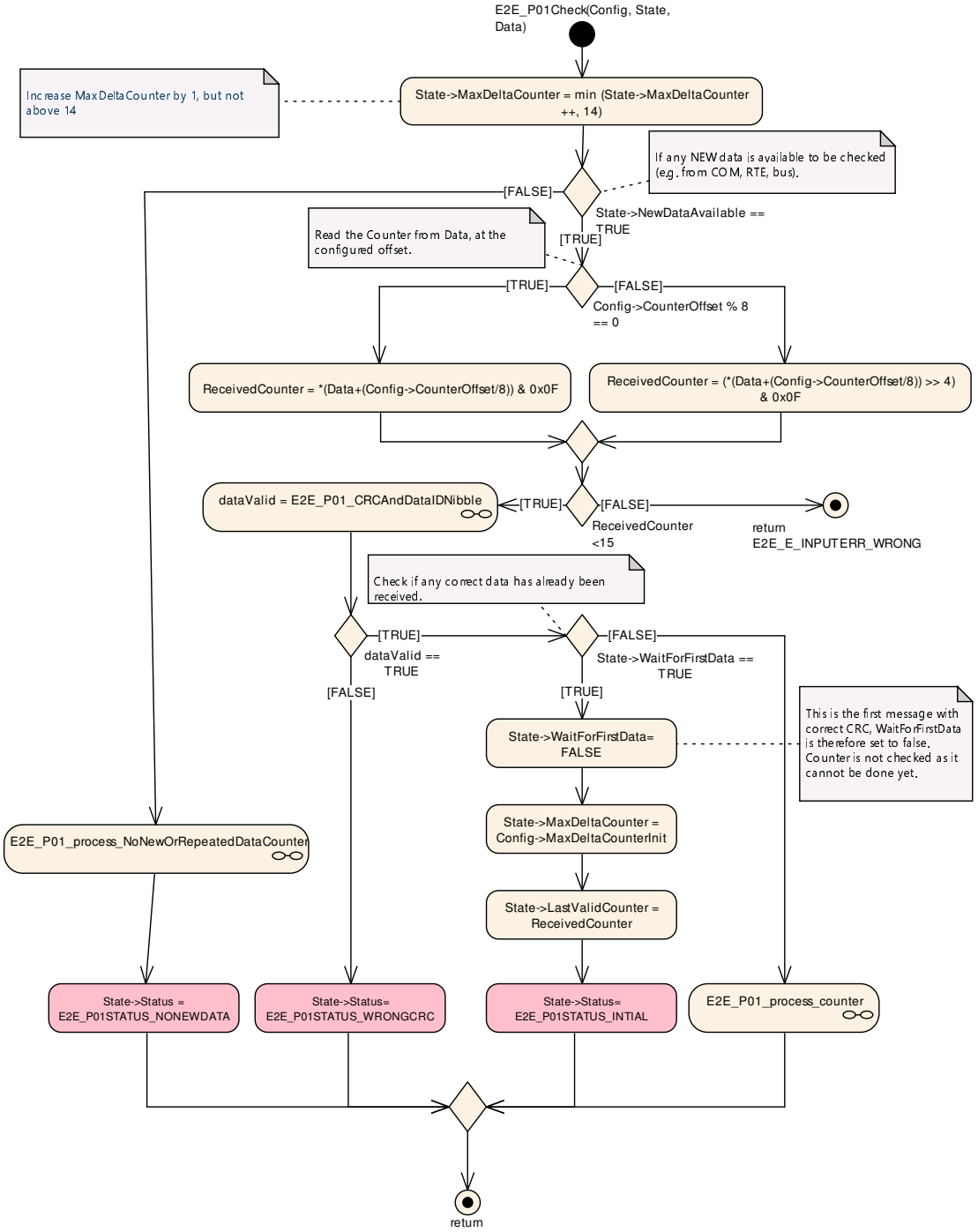

Figure 6.60: E2E_P01Check()

The diagram of the function E2E_P01Check() has a sub-diagram E2E_P01_CRCAndDataIDNibble specifying the calculation of CRC and comparing it with the received CRC, which is shown by Figure 6.57. The subroutines of Figure 6.61 are described in Figure 6.57 and Figure 6.58

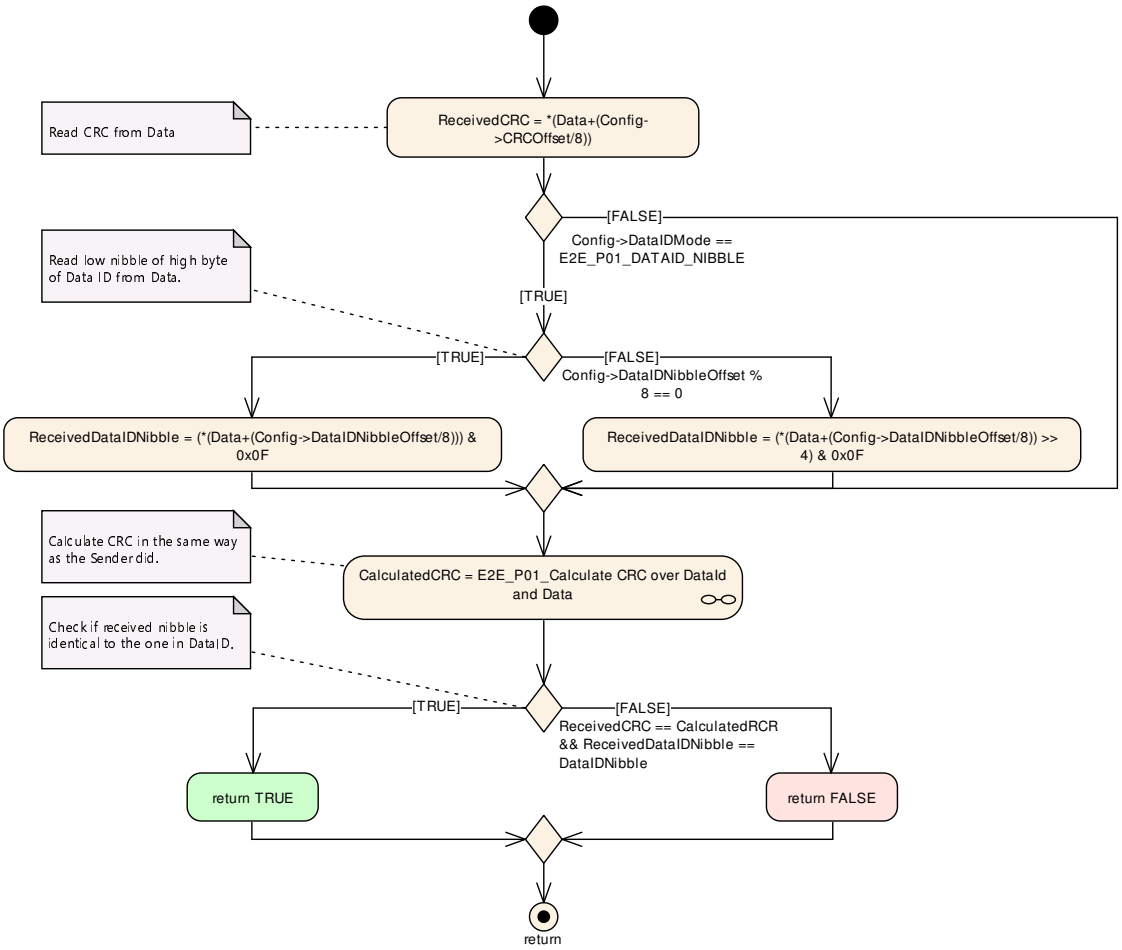

Figure 6.61: E2E Profile Check step ”E2E_P01_CRCAndDataIDNibble”

The diagram of the function E2E_P01Check() has a sub-diagram E2E_P01_process_NoNewOrRepeatedDataCounter specifying the evaluation of the different counter states, which is shown in Figure 6.62.

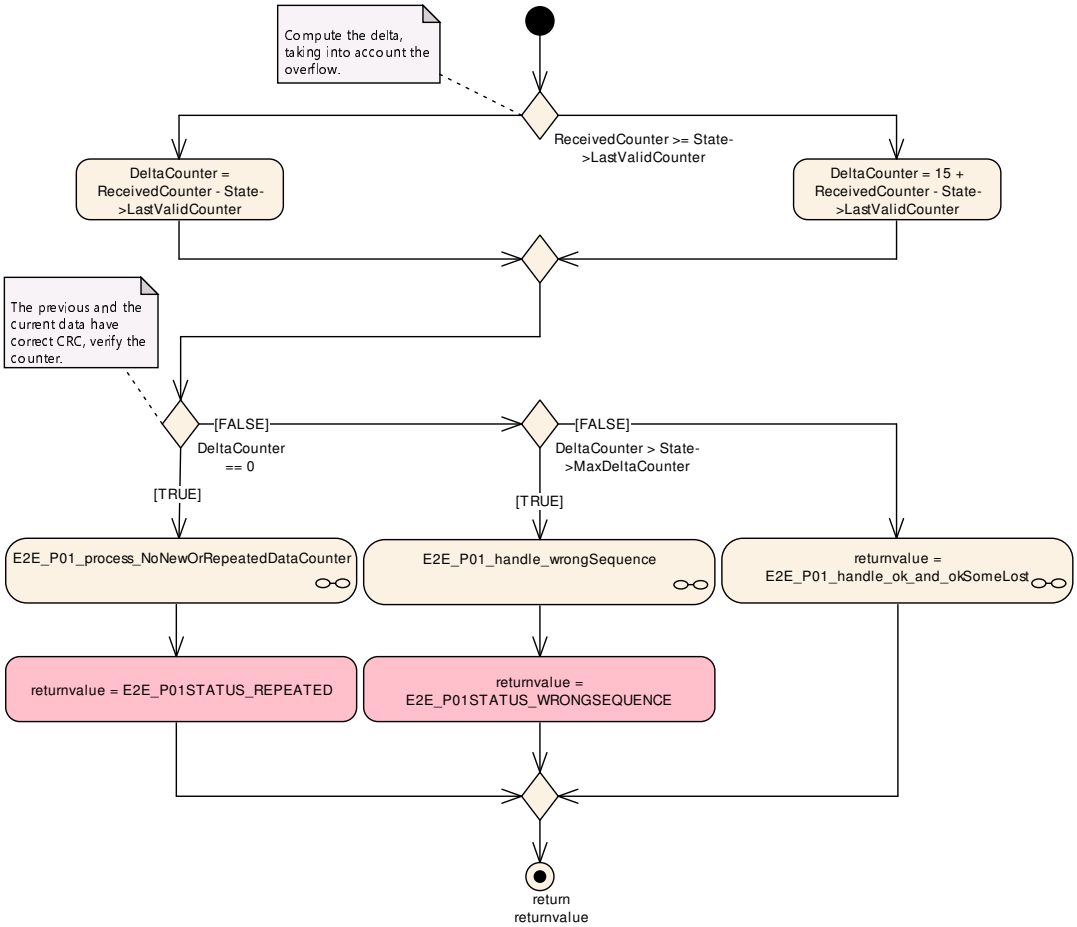

Figure 6.62: E2E Profile Check step ”E2E_P01_process_counter”

The diagram of the function E2E_P01Check() and ”E2E_P01_process_counter” have a sub-diagram E2E_P01_process_NoNewOrRepeatedDataCounter specifying the handling of receiving a repeated message and receiving no message, which is shown in Figure 6.63.

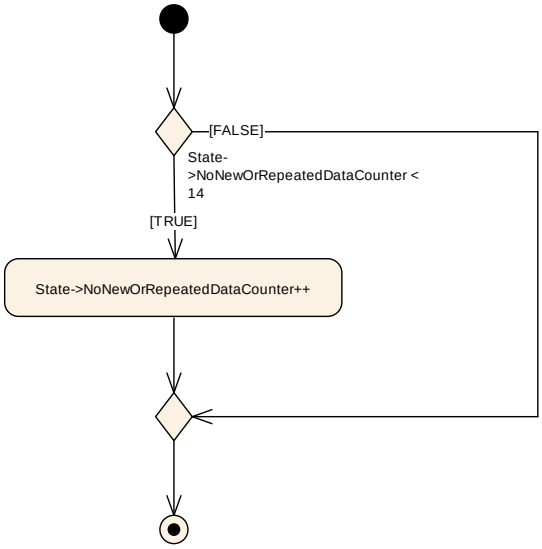

Figure 6.63: E2E Profile Check step ”E2E_P01_process_NoNewOrRepeatedDataCounter”

The diagram of the step ”E2E_P01_process_counter” has a sub-diagram ”E2E_P01_handle_wrongSequence” specifying the handling of receiving a message where the counter exceeded the maximum between two messages, which is shown in Figure 6.64.

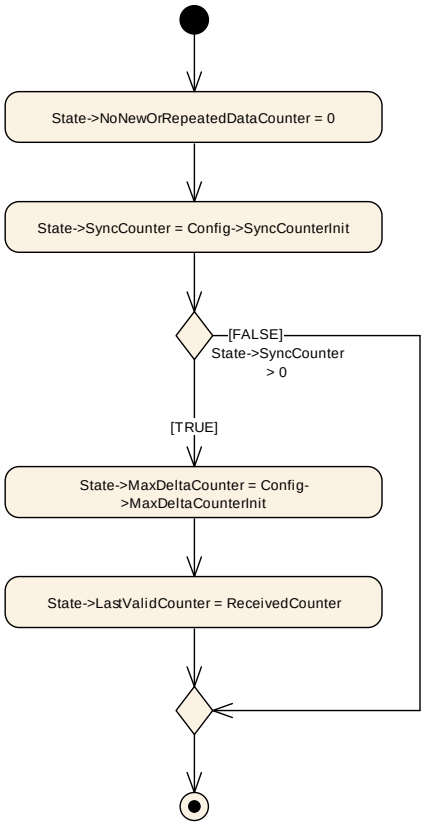

Figure 6.64: E2E Profile Check step ”E2E_P01_handle_wrongSequence”

The diagram of the step ”E2E_P01_process_counter” has a sub-diagram ”E2E_P01_handle_ok_and_okSomeLost” specifying the handling of receiving a message of valid messages where the no fault was detected, some messages where lost but this particular is valid or the the profile is synchronizing the counter, which is shown in Figure 6.65.

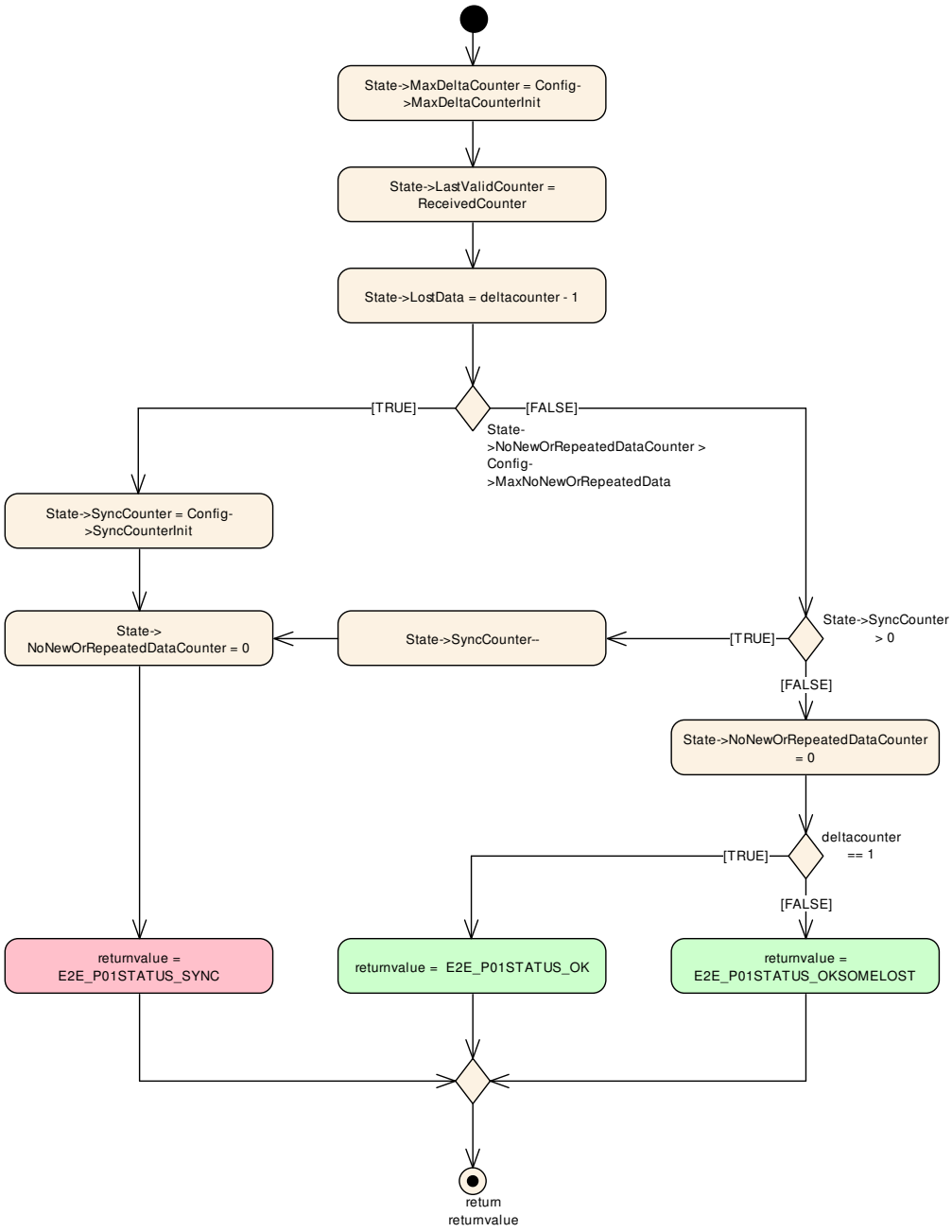

Figure 6.65: E2E Profile Check step ”E2E_P01_handle_ok_and_okSomeLost”

6.5.4 Profile Data Types

6.5.4.1 Profile 1 Protect State Type

[PRS_E2E_00644]

Upstream requirements: RS_E2E_08528

⌈The E2E_P01Protect and E2E_P01Forward functions ’state’ shall have the members defined in [PRS_E2E_00866].⌋

[PRS_E2E_00866] Profile 1 Protect State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
Counter	Unsigned Integer	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.

⌋

6.5.4.2 Profile 1 Check Status Type

[PRS_E2E_00645]

Upstream requirements: RS_E2E_08528

⌈The E2E_P01Check functions 'State' shall have the members defined in [PRS_E2E_00867].⌋

[PRS_E2E_00867] E2E Profile 1 Check Status Type Members

Upstream requirements: RS_E2E_08528

⌈

Member Name	Type	Description
LastValidCounter	Unsigned Integer	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	Unsigned Integer	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
WaitForFirstData	Boolean	If true, that means no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
NewDataAvailable	Boolean	Indicates that new data is available to be checked. This attribute has to be set by the caller of the E2E_P01Check function.

LostData	Unsigned Integer	Number of data (messages) lost since reception of last valid one. This attribute is set only if Status equals E2E_P01STATUS_OK or E2E_P01STATUS_OKSOMELOST. For other values of Status, the value of LostData is undefined.
Status	Enumeration	Result of the verification of the Data, determined by the Check function.
SyncCounter	Unsigned Integer	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.
NoNewOrRepeatedData	Unsigned Integer	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.

⌋

6.5.4.3 Profile 1 Check Status Enumeration

[PRS_E2E_00588]

Upstream requirements: RS_E2E_08528

⌈The E2E_P01Check function 'State->Status' enumeration type shall have the following enumeration values (see [PRS_E2E_00868]).⌋

[PRS_E2E_00868] E2E Profile 1 Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_P01STATUS_OK	OK	The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
E2E_P01STATUS_NONEWDATA	Error	The Check function has been invoked but no new Data is available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed

E2E_P01STATUS_WRONGCRC	Error	The data has been received according to communication medium, but 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	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	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	Error	The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
E2E_P01STATUS_OKSOMELOST	OK	The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter = MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
E2E_P01STATUS_WRONGSEQUENCE	Error	The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.

6.5.4.4 Profile 1 Configuration Type

[PRS_E2E_00646]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P01Protect, E2E_P01Forward and E2E_P01Check functions 'Config' shall have the following members defined in [PRS_E2E_00869].⌋

[PRS_E2E_00869] E2E Profile 1 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

MemberName	Type	Description
CounterOffset	Unsigned Integer	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	Unsigned Integer	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	Unsigned Integer	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	Unsigned Integer	Bit offset of the low nibble of the high byte of Data ID.
DataIDMode	Enumeration	Inclusion mode of ID in CRC computation (both bytes, alternating, or low byte only of ID included).
DataLength	Unsigned Integer	Length of data, in bits. The value shall be a multiple of 8 and shall be <= 256.
MaxDeltaCounterInit	Unsigned Integer	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	Unsigned Integer	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
SyncCounterInit	Unsigned Integer	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.

⌋

6.5.5 E2E Profile 1 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P01ConfigType field	Value
CounterOffset	8
CRCOffset	0
DataID	0x123
DataIDNibbleOffset	12
DataIDMode	E2E_P01_DATAID_BOTH
DataLength	64
MaxDeltaCounterInit	1
MaxNoNewOrRepeatedData	15
SyncCounterInit	0

Table 6.4: E2E Profile 1 protocol example configuration

E2E_P01ProtectStateType field	Value
Counter	0

Table 6.5: E2E Profile 1 example state initialization

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0xcc	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.6: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_BOTH, counter 0

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x91	0x01	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.7: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_BOTH, counter 1

6.5.5.1 DataIDMode set to E2E_P01_DATAID_ALT

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0xCE	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.8: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_ALT, counter 0

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x02	0x01	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.9: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_ALT, counter 1

6.5.5.2 DataIDMode set to E2E_P01_DATAID_LOW

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0xCE	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.10: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_LOW, counter 0

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x93	0x01	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.11: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_LOW, counter 1

6.5.5.3 DataIDMode set to E2E_P01_DATAID_NIBBLE

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0x2a	0x10	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.12: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_NIBBLE, counter 0

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x77	0x11	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.13: E2E Profile 1 protect result DataIDMode = E2E_P01_DATAID_NIBBLE, counter 1

6.6 Specification of E2E Profile 2

Profile 2 is a Legacy Profile and is only maintained for compatibility reasons. New Projects shall use Profile 22.

[PRS_E2E_00219]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548

⌈Profile 2 shall provide the following mechanisms: Sequence Number (Counter), Message Key used for CRC calculation (Data ID), Data ID + CRC, Safety Code (CRC) (see [PRS_E2E_00870]).⌋

[PRS_E2E_00870] E2E Profile 2 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548

⌈

Mechanism	Description
Counter (Sequence Number)	4bit (explicitly sent) representing numbers from 0 to 15 incremented by 1 on every send request (Bit 0:3 of Data[1]) at sender side. The counter is incremented on every call of the E2E_P02Protect() function, i.e. on every transmission request of the SW-C
Data ID (Message Key used for CRC calculation)	8 bit (not explicitly sent) The specific Data ID used to calculate the CRC depends on the value of the Counter and is an element of an pre-defined set of Data IDs (value of the counter as index to select the particular Data ID used for the protection). For every Data element, the List of Data IDs depending on each value of the counter is unique.
Data ID + CRC	Masquerade and incorrect addressing, insertion
Safety Code(CRC(Safety Code))	8 bit explicitly sent (Data[0]) Polynomial: 0x2F (x8 + x5 + x3 + x2 + x + 1) Start value: 0xFF Final XOR-value: 0xFF Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay and CAN.
Timeout monitoring	Timeout is determined by E2E Supervision by means of evaluation of the Counter, by a nonblocking read at the receiver. Timeout is reported by E2E Supervision to the caller by means of the status flags in E2E_P02CheckStatusType.

⌋

The mechanisms provided by Profile 2 enable the detection of the relevant failure modes except message delay (for details see the table in [PRS_E2E_00870]):

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

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

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

Table 6.14: Detectable communication faults using Profile 2

6.6.1 Header Layout

[PRS_E2E_00121]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 2, the layout of the data buffer (Data) shall be as depicted in Figure 6.66.⌋



Figure 6.66: E2E Profile 2 data buffer layout

6.6.1.1 Counter

[PRS_E2E_00123]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00128]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00129]

Upstream requirements: RS_E2E_08528

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

[3] Implementation by sender and receiver
[4] for a set of data protected by same CRC

6.6.1.2 DataID

[PRS_E2E_00119]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00120]

Upstream requirements: RS_E2E_08528

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

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

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

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

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

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

6.6.1.3 CRC

[PRS_E2E_00117]

Upstream requirements: RS_E2E_08528

⌈E2E Profile 2 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Supervision for calculating CRC checksums.⌋

[PRS_E2E_00118]

Upstream requirements: RS_E2E_08528

⌈E2E Profile 2 shall use 0xFF as the start value CRC_StartValue8 for CRC calculation.⌋

[PRS_E2E_00122]

Upstream requirements: RS_E2E_08528

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

6.6.2 Creation of E2E-Header

[PRS_E2E_00124]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 2, both the E2E_P02Protect() and the E2E_P02Forward() function shall not modify any bit of Data except the bits representing the CRC and the Counter.⌋

6.6.2.1 E2E_P02Protect

The E2E_P02Protect() function of E2E Profile 2 is called by a SW-C in order to protect its application data against the failure modes as shown in table in [PRS_E2E_00870]. E2E_P02Protect() therefore calculates the Counter and then the CRC and puts both into the data buffer (Data). A flow chart with the visual description of the function E2E_P02Protect() is depicted in Figure 6.67 and Figure 6.68.

[PRS_E2E_00126]

Upstream requirements: RS_E2E_08528, RS_E2E_08531

⌈In E2E Profile 2, the E2E_P02Protect() function shall perform the activities as specified in Figure 6.67 and Figure 6.68.⌋

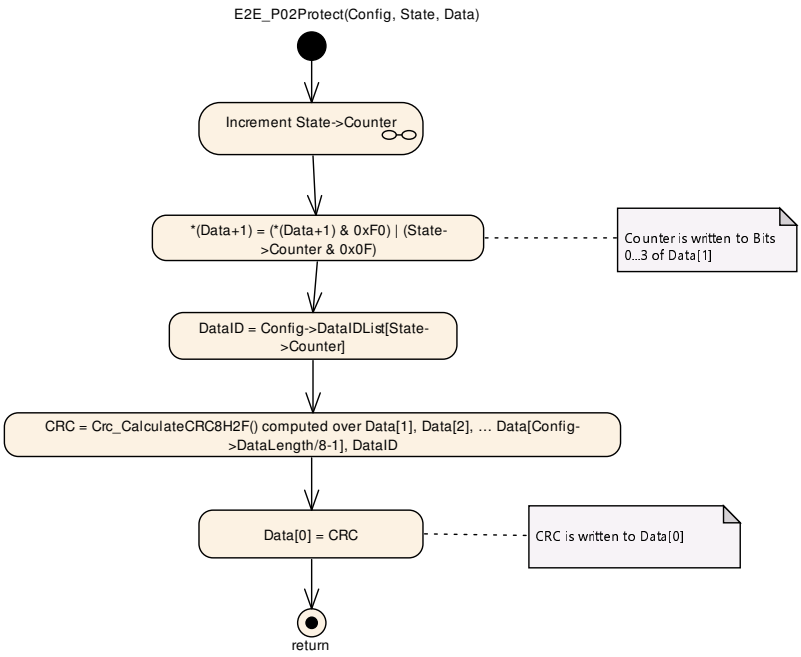

Figure 6.67: E2E_P02Protect()

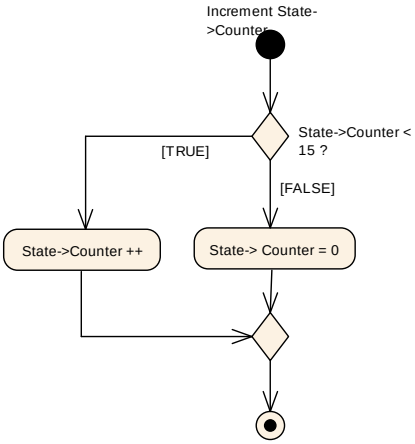

Figure 6.68: Increment Counter

[PRS_E2E_00127]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00130]

Upstream requirements: RS_E2E_08528

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

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

[PRS_E2E_00132]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 2, after determining the specific Data ID, the E2E_P02Protect() and E2E_P02Forward() functions shall calculate the CRC over Data[1], Data[2], ... Data[Config->DataLength/8-1] of the data buffer (Data) extended with the Data ID.⌋

[PRS_E2E_00133]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 2, the E2E_P02Protect() and E2E_P02Forward() functions shall update the CRC (i.e. Data[0]) in the data buffer (Data) after computing the CRC.⌋

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

6.6.2.2 E2E_P02Forward

The E2E_P02Forward() function of E2E Profile 2 is called by a SW-C in order to protect its application data and forward a received E2E-Status for use cases like translation of signal based to service oriented communication. If the received E2E status equals E2E_P_OK the behavior of the function shall be the same like E2E_P02Protect(). A flow chart with the visual description of the function E2E_P02Forward() is depicted in Figure 6.69 and Figure 6.70.

[PRS_E2E_00613]

Upstream requirements: RS_E2E_08528, RS_E2E_08531

⌈In E2E Profile 2, the E2E_P02Forward() function shall perform the activities as specified in Figure 6.69 and Figure 6.68.⌋

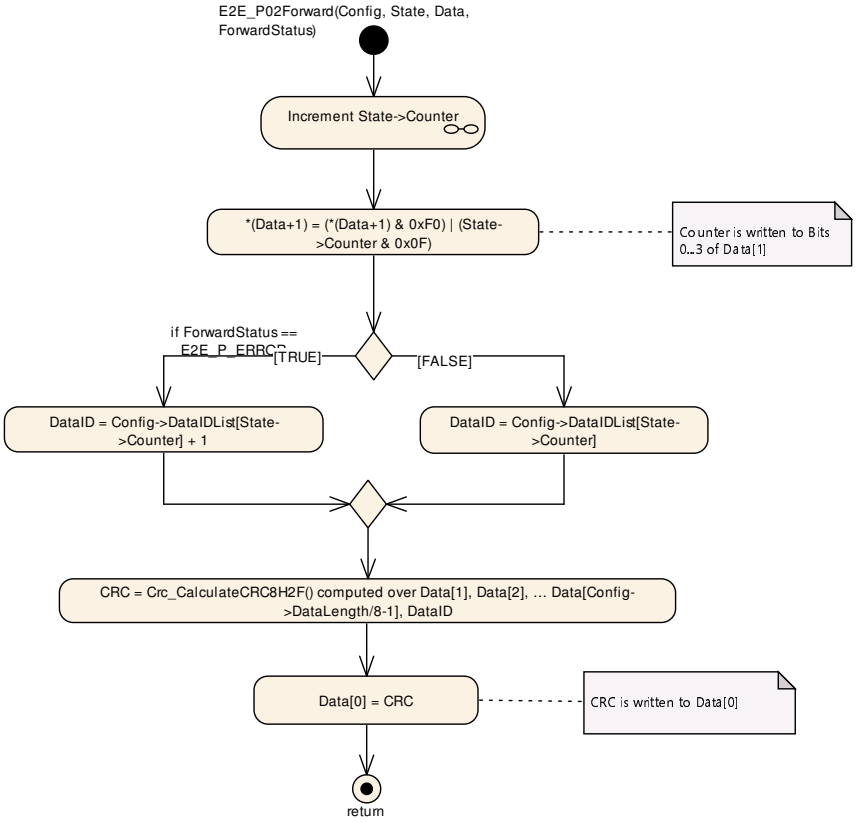

Figure 6.69: E2E_P02Forward()

[PRS_E2E_00614]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 2, the E2E_P02Forward() function shall increment the Counter according to Figure 6.70.⌋

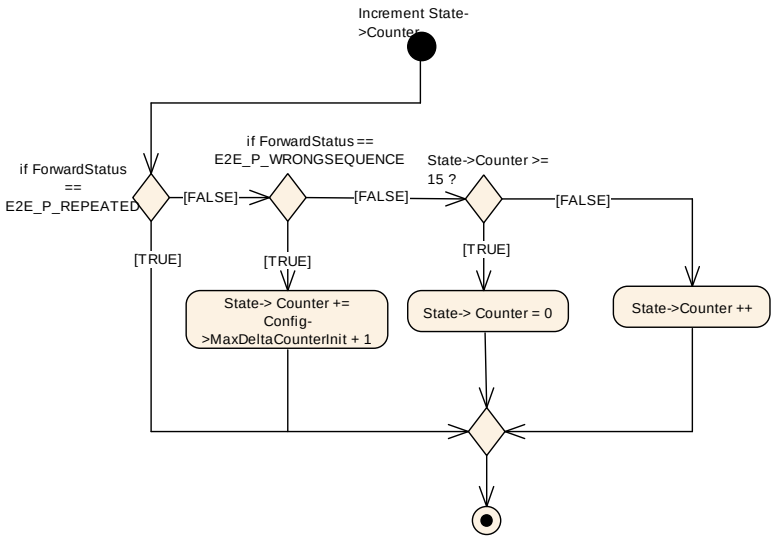

Figure 6.70: Increment Counter

6.6.3 Evaluation of the E2E-Check

[PRS_E2E_00125]

Upstream requirements: RS_E2E_08528, RS_E2E_08531

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

6.6.3.1 E2E_P02Check

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

A flow chart with the visual description of the function E2E_P02Check() is depicted in Figure 6.71, Figure 6.72 and Figure 6.73.

[PRS_E2E_00134]

Upstream requirements: RS_E2E_08528, RS_E2E_08531

⌈In E2E Profile 2, the E2E_P02Check() function shall perform the activities as specified in Figure 6.71, Figure 6.72 and Figure 6.73.⌋

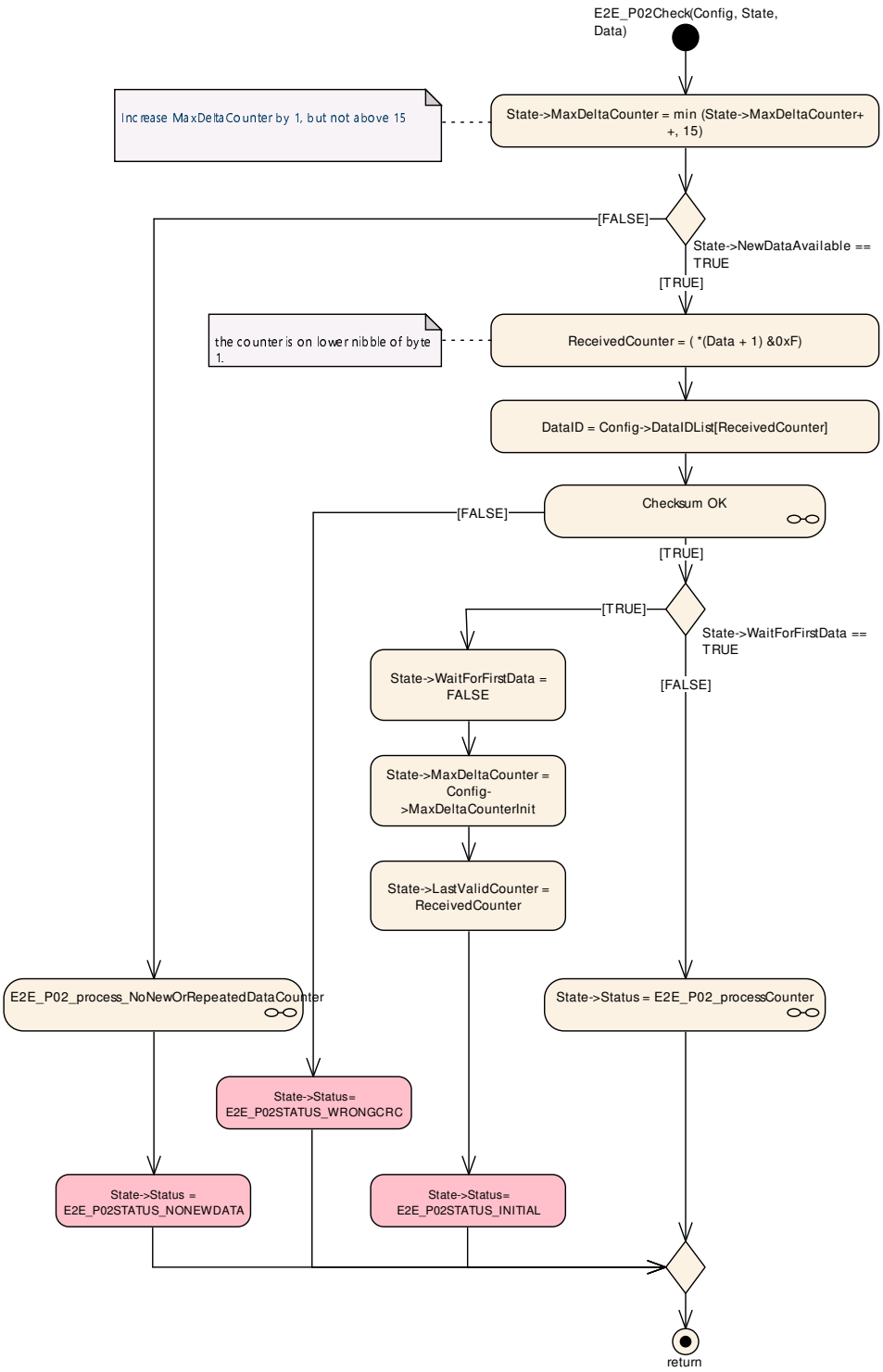

Figure 6.71: E2E_P02Check

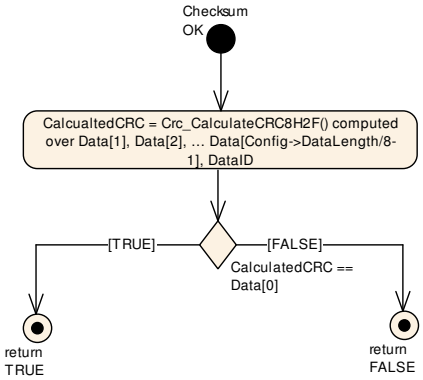

Figure 6.72: Checksum OK

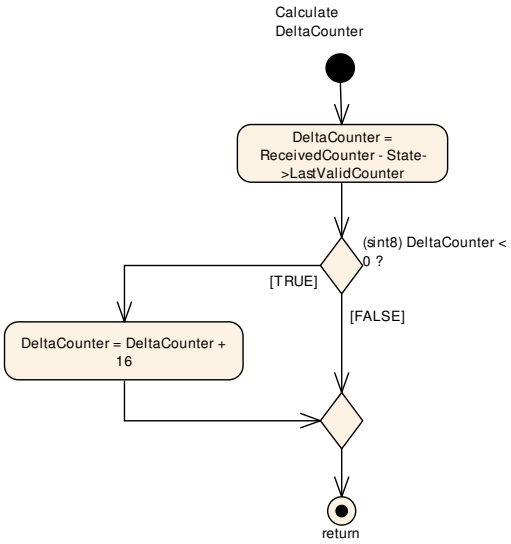

Figure 6.73: Calculate Delta Counter

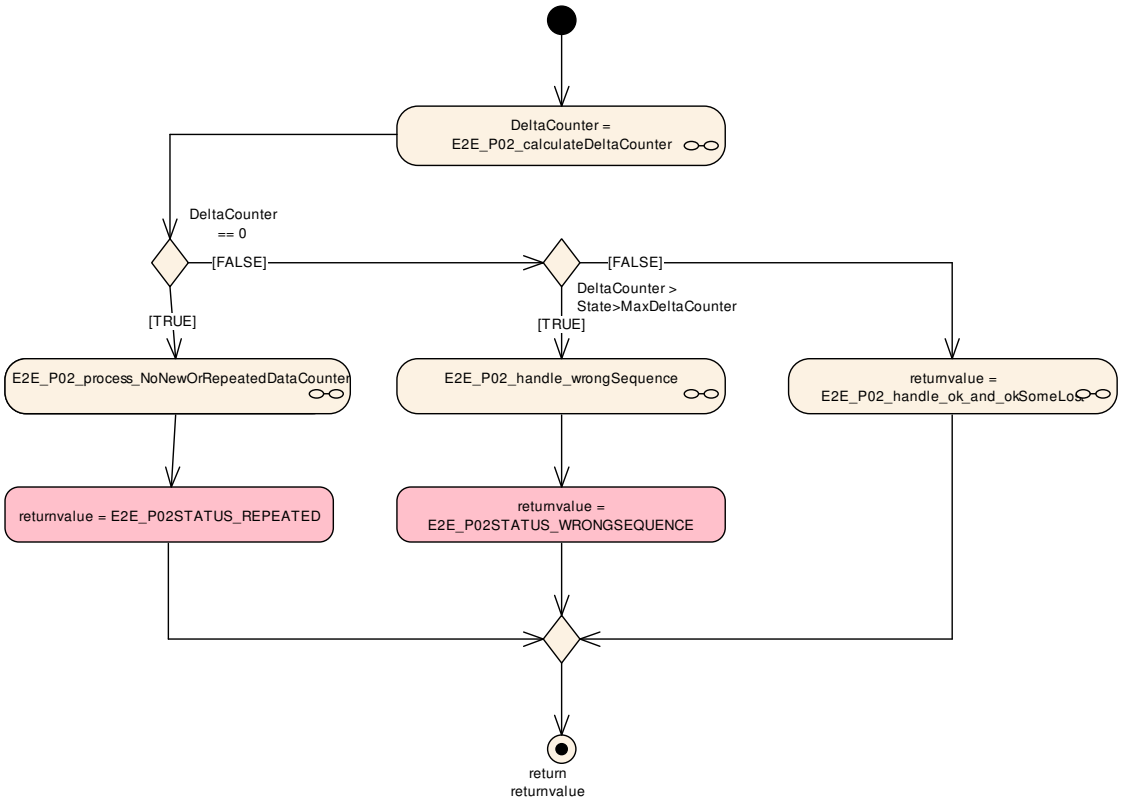

Figure 6.74: E2E Profile Check step ”E2E_P02_process_counter”

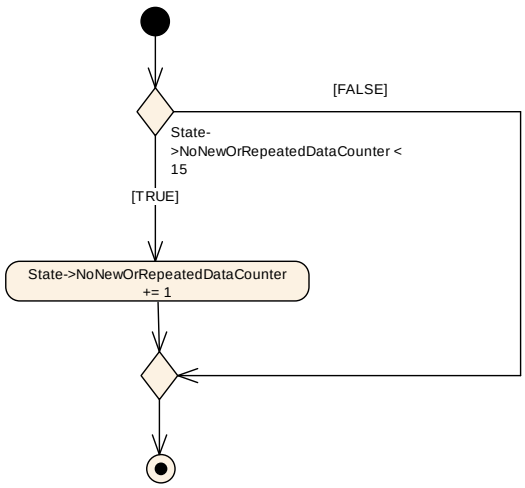

Figure 6.75: E2E Profile Check step ”E2E_P02_process_NoNewOrRepeatedDataCounter”

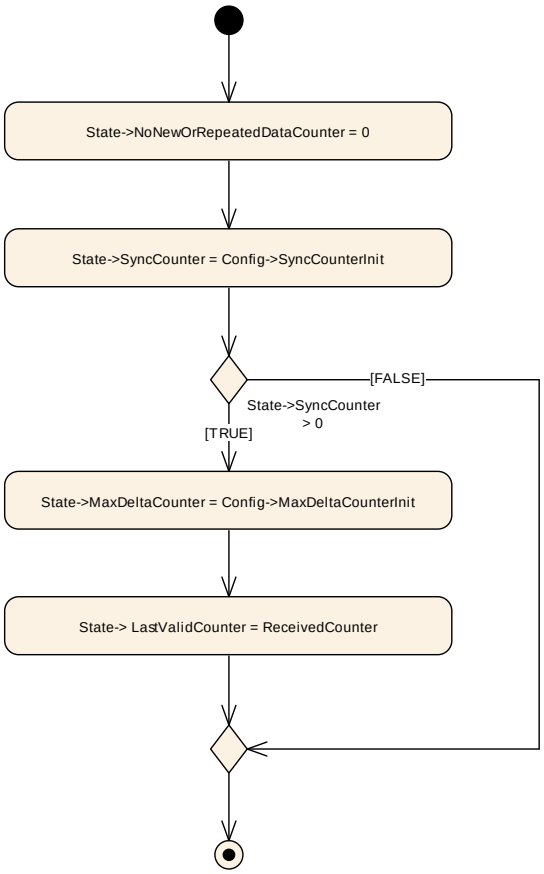

Figure 6.76: E2E Profile Check step ”E2E_P02_handle_wrongSequence”

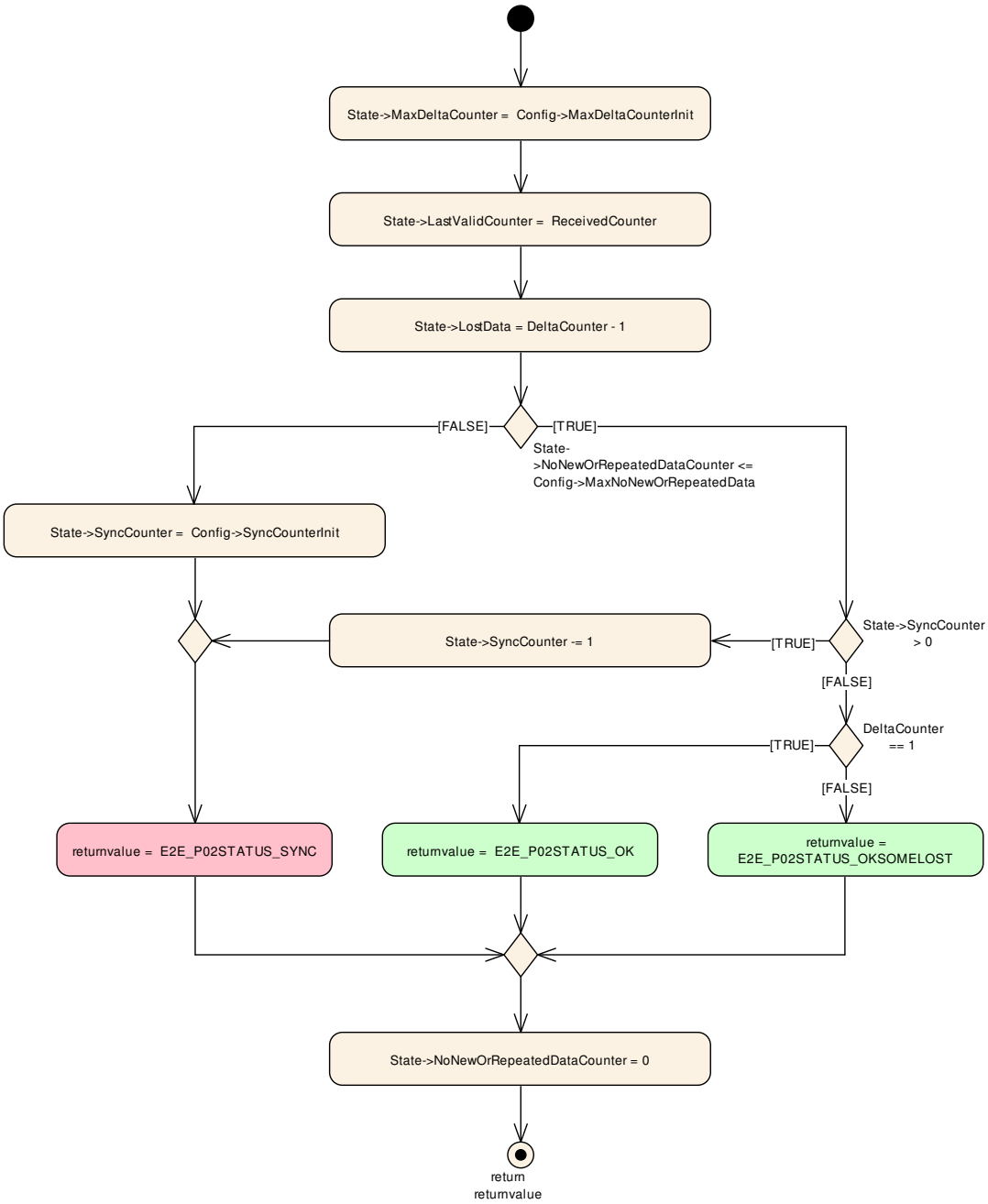

Figure 6.77: E2E Profile Check step ”E2E_P02_handle_ok_and_okSomeLost”

First, the E2E_P02Check() function increments the value MaxDeltaCounter by 1 up to maximum value 15. MaxDeltaCounter specifies the maximum allowed difference between two Counter values of two consecutively received valid messages. Note: MaxDeltaCounter is used in order to perform a plausibility check for the failure mode re-sequencing. If the flag NewDataAvailable is set, the E2E_P02Check() function continues with the evaluation of the CRC. Otherwise, it returns with Status set to E2E_P02STATUS_NONEWDATA. To evaluate the correctness of the CRC, the following actions are performed:

- The specific Data ID is determined using the value of the Counter as provided in Data.
- Then the CRC is calculated over Data payload extended with the Data ID as last Byte: CalculatedCRC = Crc_CalculateCRC8H2F() calculated over Data[1], Data[2], ... Data[Config->DataLength/8-1], Data ID
- Finally, the check for correctness of the received Data is performed by comparing CalculatedCRC with the value of CRC stored in Data.

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

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

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

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

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

[PRS_E2E_00135]

Upstream requirements: RS_E2E_08528

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

Details on the calculation of DeltaCounter are depicted in Figure 6.74.

[PRS_E2E_00136]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00137]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00138]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00139]

Upstream requirements: RS_E2E_08528

⌈In E2E Profile 2, the E2E_P02Check() function shall determine the specific Data ID from DataIDList using the Counter of the received Data as index.⌋

[PRS_E2E_00140]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00141]

Upstream requirements: RS_E2E_08528

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

⌋

[PRS_E2E_00142]

Upstream requirements: RS_E2E_08528

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

[PRS_E2E_00143]

Upstream requirements: RS_E2E_08528

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

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

[PRS_E2E_00145]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall

- set Status to E2E_P02STATUS_WRONGSEQUENCE; and
- re-initialize SyncCounter with SyncCounterInit

if the calculated value of DeltaCounter exceeds the value of MaxDeltaCounter.⌋

[PRS_E2E_00146]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall set Status to E2E_P02STATUS_REPEATED if the calculated DeltaCounter equals 0.⌋

[PRS_E2E_00147]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall set Status to E2E_P02STATUS_OK if the following conditions are true:

- the calculated DeltaCounter equals 1; and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State –> NoNewOrRepeatedDataCounter <= Config –> MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0.

⌋

[PRS_E2E_00298]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall

- re-initialize SyncCounter with SyncCounterInit; and
- set Status to E2E_P02STATUS_SYNC; if the following conditions are true:
- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. 1 =/< DeltaCounter =/< MaxDeltaCounter); and
- the value of the NoNewOrRepeatedDataCounter exceeds MaxNoNewOrRepeatedData. (i.e. State NoNewOrRepeatedDataCounter > Config MaxNoNewOrRepeatedData)

⌋

[PRS_E2E_00299]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall

- decrement SyncCounter by 1 and set Status to E2E_P02STATUS_SYNC if the following conditions are true:
- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. 1 =/< DeltaCounter =/< MaxDeltaCounter); and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State NoNewOrRepeatedDataCounter =/< Config MaxNoNewOrRepeatedData); and
- the SyncCounter exceeds 0.

⌋

[PRS_E2E_00148]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall set Status to E2E_P02STATUS_OKSOMELOST if the following conditions are true:

- the calculated DeltaCounter is greater-than 1 but less-than or equal to MaxDeltaCounter (i.e. 1 < DeltaCounter =/< MaxDeltaCounter); and
- the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State NoNewOrRepeatedDataCounter =/< Config MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0.

⌋

[PRS_E2E_00149]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall set the value LostData to (DeltaCounter - 1) if the calculated DeltaCounter is greater-than 1 but less-than or equal to MaxDeltaCounter.⌋

[PRS_E2E_00150]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall re-initialize MaxDeltaCounter with MaxDeltaCounterInit if it returns one of the following Status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_INITIAL; or

- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE on condition that SyncCounter exceeds 0 (i.e. SyncCounter > 0).

⌋

[PRS_E2E_00151]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall set LastValidCounter to Counter of Data if it returns one of the following Status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_INITIAL; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE on condition that SyncCounter exceeds 0 (i.e. SyncCounter > 0).

⌋

[PRS_E2E_00300]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check() function shall reset the NoNewOrRepeatedDataCounter to 0 if it returns one of the following status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE

⌋

[PRS_E2E_00301]

Upstream requirements: RS_E2E_08528

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

6.6.4 Profile Data Types

6.6.4.1 Profile 2 Protect State Type

[PRS_E2E_00647]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Protect and E2E_P02Forward functions 'state' shall have the members defined in [PRS_E2E_00871].⌋

[PRS_E2E_00871] E2E Profile 2 Protect State Type

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
Counter	Unsigned Integer	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

⌋

6.6.4.2 Profile 2 Check Status Type

[PRS_E2E_00648]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P02Check functions 'State' shall have the members defined in [PRS_E2E_00872].⌋

[PRS_E2E_00872] E2E Profile 2 Check Status Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
LastValidCounter	Unsigned Integer	Counter of last valid received message.
MaxDeltaCounter	Unsigned Integer	MaxDeltaCounter specifies the maximum allowed difference between two counter values of consecutively received valid messages.
WaitForFirstData	Boolean	If true, that means no correct data (with correct Data ID and CRC) has been yet received after the receiver initialization or reinitialization.
NewDataAvailable	Boolean	Indicates that a new data is available to be checked. This attribute is set by the E2E_P02Check function caller, and not by the function itself.

LostData	Unsigned Integer	Number of data (messages) lost since reception of last valid one.
Status	Enumeration	Result of the verification of the Data, determined by the Check function.
SyncCounter	Unsigned Integer	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.
NoNewOrRepeatedData	Unsigned Integer	Amount of consecutive reception cycles in which either (1) there was no new data, or (2) when the data was repeated.

⌋

6.6.4.3 Profile 2 Check Status Enumeration

[PRS_E2E_00589]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Check functions 'State->Status' enumeration type shall consist of the following enumeration values (see [PRS_E2E_00873]).⌋

[PRS_E2E_00873] E2E Profile 2 Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_P02STATUS_OK	OK	The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by 1 with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that no Data has been lost since the last correct data reception.
E2E_P02STATUS_NONEWDATA	Error	The Check function has been invoked but no new Data is available since the last call, according to communication medium (e.g. RTE, COM). As a result, no E2E checks of Data have been consequently executed.
E2E_P02STATUS_WRONGCRC	Error	The data has been received according to communication medium, but the CRC is incorrect.

E2E_P02STATUS_SYNC	Not Valid	The new data has been received after detection of an unexpected behavior of counter. The data has a correct CRC and a counter within the expected range with respect to the most recent Data received, but the determined continuity check for the counter is not finalized yet.
E2E_P02STATUS_INITIAL	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	Error	The new data has been received according to communication medium, the CRC is correct, but the Counter is identical to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST.
E2E_P02STATUS_OKSOMELOST	OK	The new data has been received according to communication medium, the CRC is correct, the Counter is incremented by DeltaCounter (1 < DeltaCounter =MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that some Data in the sequence have been probably lost since the last correct/initial reception, but this is within the configured tolerance range.
E2E_P02STATUS_WRONGSEQUENCE	Error	The new data has been received according to communication medium, the CRC is correct, but the Counter Delta is too big (DeltaCounter > MaxDeltaCounter) with respect to the most recent Data received with Status _INITIAL, _OK, or _OKSOMELOST. This means that too many Data in the sequence have been probably lost since the last correct/initial reception.

⌋

6.6.4.4 Profile 2 Configuration Type

[PRS_E2E_00667]

Upstream requirements: RS_E2E_08528

⌈The E2E_P02Protect, E2E_P02Forward and E2E_P02Check functions 'Config' shall have the following members defined in [PRS_E2E_00874].⌋

[PRS_E2E_00874] E2E Profile 2 Configuration Type

Upstream requirements: RS_E2E_08528

⌈

Member Name	Type	Description
DataLength	Unsigned Integer	Length of Data, in bits. The value shall be a multiple of 8.
DataIDList	Unsigned Integer Array	An array of appropriately chosen Data IDs for protection against masquerading.
MaxDeltaCounterInit	Unsigned Integer	Initial maximum allowed gap between two counter values of two consecutively received valid Data.
MaxNoNewOrRepeatedData	Unsigned Integer	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
SyncCounterInit	Unsigned Integer	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	Unsigned Integer	Offset of the E2E header in the Data[] array in bits. It shall be: 0 <= Offset <= DataLength-(2*8).Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1.

⌋

6.6.5 E2E Profile 2 Protocol Examples

E2E_P02ConfigType field	Value
DataLength	64
DataIDList	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10
MaxDeltaCounterInit	1
MaxNoNewOrRepeatedData	15
SyncCounterInit	0
Offset	0

Table 6.15: E2E Profile 2 protocol example configuration

E2E_P02ProtectStateType field	Value
Counter	0

Table 6.16: E2E Profile 2 example state initialization

Result data of E2E_P02Protect() with data equals all zeros (0x00), counter starting with 1 (note: first used counter is 1, although counter field is initialized with 0, as counter is incremented before usage):

Counter	DataID	Byte							
		0	1	2	3	4	5	6	7
1	0x02	0x1b	0x01	0x00	0x00	0x00	0x00	0x00	0x00
2	0x03	0x98	0x02	0x00	0x00	0x00	0x00	0x00	0x00
3	0x04	0x31	0x03	0x00	0x00	0x00	0x00	0x00	0x00
4	0x05	0x0d	0x04	0x00	0x00	0x00	0x00	0x00	0x00
5	0x06	0x18	0x05	0x00	0x00	0x00	0x00	0x00	0x00
6	0x07	0x9b	0x06	0x00	0x00	0x00	0x00	0x00	0x00
7	0x08	0x65	0x07	0x00	0x00	0x00	0x00	0x00	0x00
8	0x09	0x08	0x08	0x00	0x00	0x00	0x00	0x00	0x00
9	0x0a	0x1d	0x09	0x00	0x00	0x00	0x00	0x00	0x00
10	0x0b	0x9e	0x0a	0x00	0x00	0x00	0x00	0x00	0x00
11	0x0c	0x37	0x0b	0x00	0x00	0x00	0x00	0x00	0x00
12	0x0d	0x0b	0x0c	0x00	0x00	0x00	0x00	0x00	0x00
13	0x0e	0x1e	0x0d	0x00	0x00	0x00	0x00	0x00	0x00
14	0x0f	0x9d	0x0e	0x00	0x00	0x00	0x00	0x00	0x00
15	0x10	0xcd	0x0f	0x00	0x00	0x00	0x00	0x00	0x00
0	0x01	0x0e	0x00	0x00	0x00	0x00	0x00	0x00	0x00
		CRC	*4 bit Data + 4 bit Counter*	*Data*					

Table 6.17: E2E Profile 2 example protect result

6.7 Specification of E2E Profile 4

[PRS_E2E_00372]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549, RS_E2E_08550

⌈Profile 4 shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID (see [PRS_E2E_00875]).⌋

[PRS_E2E_00875] E2E Profile 4 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549, RS_E2E_08550

⌈

Control field	Description
Length	16 bits, to support dynamic-size data.
Counter	16-bits.

CRC	32 bits, polynomial in normal form 0xF4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCP/IP.
Data ID	32-bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P04LENGTH_POS	0
P04LENGTH_LEN	2
P04COUNTER_POS	2
P04COUNTER_LEN	2
P04DATAID_POS	4
P04DATAID_LEN	4
P04CRC_POS	8
P04CRC_LEN	4
P04CALCULATE_CRC	Crc_CalculateCRC32P4()

Table 6.18: Profile 4-specific data

For behavior and flowcharts of E2E Profile 04 see Chapter 6.3.

6.7.1 Header Layout

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

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

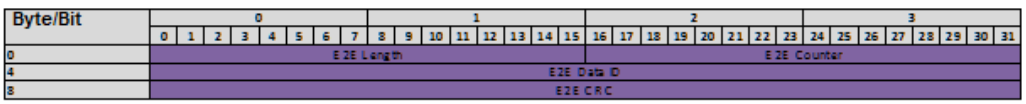

Figure 6.78: E2E Profile 4 Header

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

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

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

6.7.2 Profile 4 Configuration Type

[PRS_E2E_00651]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P04Protect, E2E_P04Forward and E2E_P04Check functions 'Config' shall have the following members defined in [PRS_E2E_00876].⌋

[PRS_E2E_00876] E2E Profile 4 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be <= MaxDataLength and shall be >= 12*8. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 4096*8 (4KB) and it shall be >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.7.3 E2E Profile 4 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P04ConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000
MinDataLength	96
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.19: E2E Profile 4 protocol example configuration

E2E_P04ProtectStateType field	Value
Counter	0

Table 6.20: E2E Profile 4 example state initialization

Result data of E2E_P04Protect() with short data length (length 16 bytes, means 4 actual data bytes), offset = 0, counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x10	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	8	9	10	11	12	13	14	15
Data	0x86	0x2b	0x05	0x56	0x00	0x00	0x00	0x00
Field	*CRC*				*Data*			

Table 6.21: E2E Profile 4 example short

Result data of E2E_P04Protect() with minimum data length (4 data bytes), offset = 64 (as with SOME/IP header use case), datalength = 24, counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	8	9	10	11	12	13	14	15
Data	0x00	0x18	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	16	17	18	19	20	21	22	23
Data	0x69	0xd7	0x50	0x2e	0x00	0x00	0x00	0x00
Field	*CRC*				*Data*			

Table 6.22: E2E Profile 4 example short with SOME/IP use case

6.8 Specification of E2E Profile 5

[PRS_E2E_00394]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈Profile 5 shall provide the following control fields, transmitted at runtime together with the protected data: Counter, CRC, Data ID (see [PRS_E2E_00877]).⌋

[PRS_E2E_00877] E2E Profile 5 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈

Control field	Description
Counter	8 bits. (explicitly sent)
CRC	16 bits, polynomial in normal form 0x1021 (Autosar notation), provided by CRC library. (explicitly sent)
Data ID	16 bits, unique system-wide. (implicitly sent)E2E

⌋

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

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

Table 6.23: Detectable communication faults using Profile 5

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P05LENGTH_POS	0
P05LENGTH_LEN	0
P05COUNTER_POS	2
P05COUNTER_LEN	1
P05DATAID_POS	0
P05DATAID_LEN	0
P05CRC_POS	0
P05CRC_LEN	2
P05CALCULATE_CRC	Crc_CalculateCRC16()

Table 6.24: Profile 5-specific data

6.8.1 Header layout

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

The header of the E2E Profile 5 has one fixed layout, as follows:

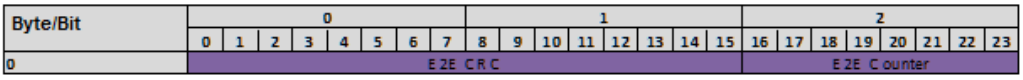

Figure 6.79: E2E Profile 5 header

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

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

6.8.1.1 Counter

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

[PRS_E2E_00397]

Upstream requirements: RS_E2E_08539

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

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

6.8.1.2 Data ID

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

[PRS_E2E_00399]

Upstream requirements: RS_E2E_08539

⌈In the E2E Profile 5, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation.⌋

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

[PRS_E2E_01441] Uniqueness of Data IDs

Replaces: PRS_E2E_UC_00463
Upstream requirements: RS_E2E_08539

⌈In the E2E profile 5, the Data IDs should be globally unique within the network of communicating system (made of several ECUs each sending different data).⌋

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

In case of usage of E2E Supervision for protecting messages (i.e. invocation from COM), the receiver COM expects at a reception only a specific message, which is checked by E2E Supervision using Data ID.

6.8.1.3 Length

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

6.8.1.4 CRC

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

[PRS_E2E_00400]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E Profile 5 shall use the Crc_CalculateCRC16() function of the SWS CRC Supervision for calculating the CRC (Polynomial: 0x1021; Autosar notation).⌋

[PRS_E2E_00401]

Upstream requirements: RS_E2E_08539, RS_E2E_08531

⌈In E2E Profile 5, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended at the end with the Data ID.⌋

6.8.2 Creation of the E2E-Header

6.8.2.1 E2E_P05Protect

The function E2E_P05Protect() performs the steps as specified by the following diagrams (see also Chapter 6.3).

[PRS_E2E_00403]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P05Protect() shall have the overall behavior as shown in Figure 6.80.⌋

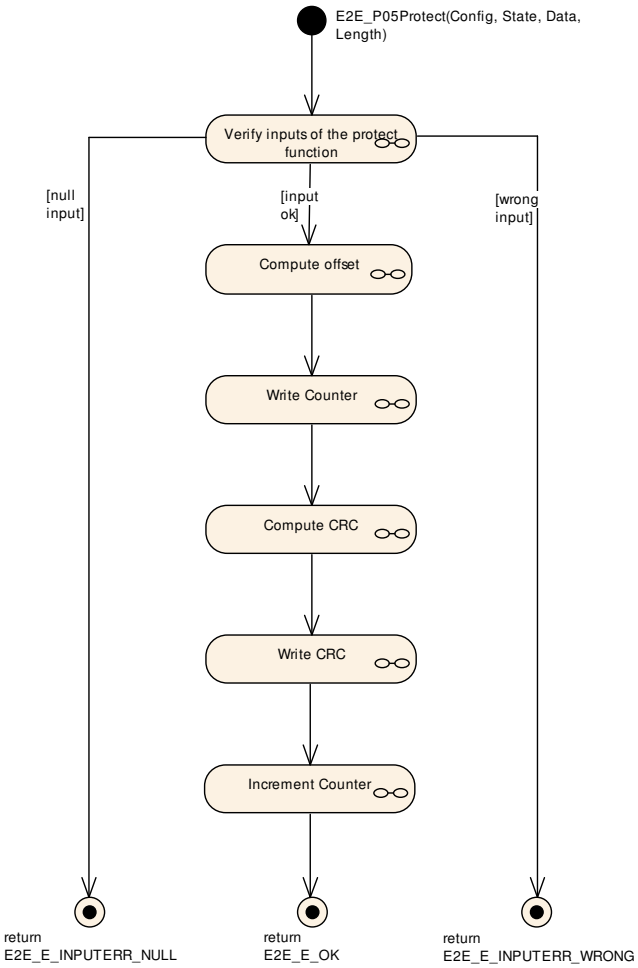

Figure 6.80: E2E Profile 5 Protect

[PRS_E2E_00404]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_P05Protect() shall behave as shown in Figure 6.3.⌋

[PRS_E2E_00469]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute offset” in E2E_P05Protect(), E2E_P05Forward() and E2E_P05Check() shall behave as shown in Figure 6.4.⌋

[PRS_E2E_00405]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P05Protect() shall behave as shown in Figure 6.6.⌋

[PRS_E2E_00406]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P05Protect() and in E2E_P05Check shall behave as shown in Figure 6.81.⌋

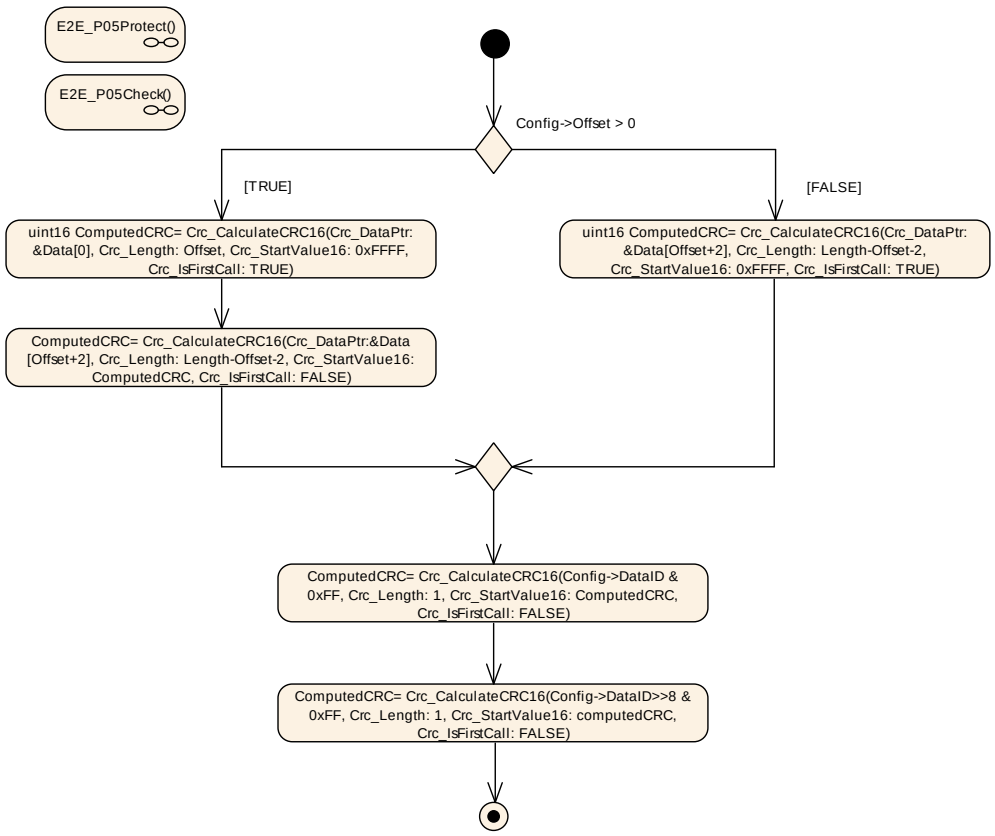

Figure 6.81: E2E Profile 5 Protect and Check step ”Compute CRC”

[PRS_E2E_00407]

Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC” in E2E_P05Protect() and E2E_P05Forward() shall behave as shown in Figure 6.9.⌋

[PRS_E2E_00409]

Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_P05Protect() and E2E_P05Forward() shall behave as shown in Figure 6.10.⌋

6.8.2.2 E2E_P05Forward

The E2E_P05Forward() function of E2E Profile 5 is called by a SW-C in order to protect its application data and forward an received E2E-Status for use cases like translation of signal based to service oriented communication. If the received E2E status equals E2E_P_OK the behavior of the function shall be the same like E2E_P05Protect(). The function E2E_P05Forward() performs the steps as specified by the following diagrams (see also Chapter 6.3).

[PRS_E2E_00639]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P05Forward() shall have the overall behavior as shown in Figure 6.82.⌋

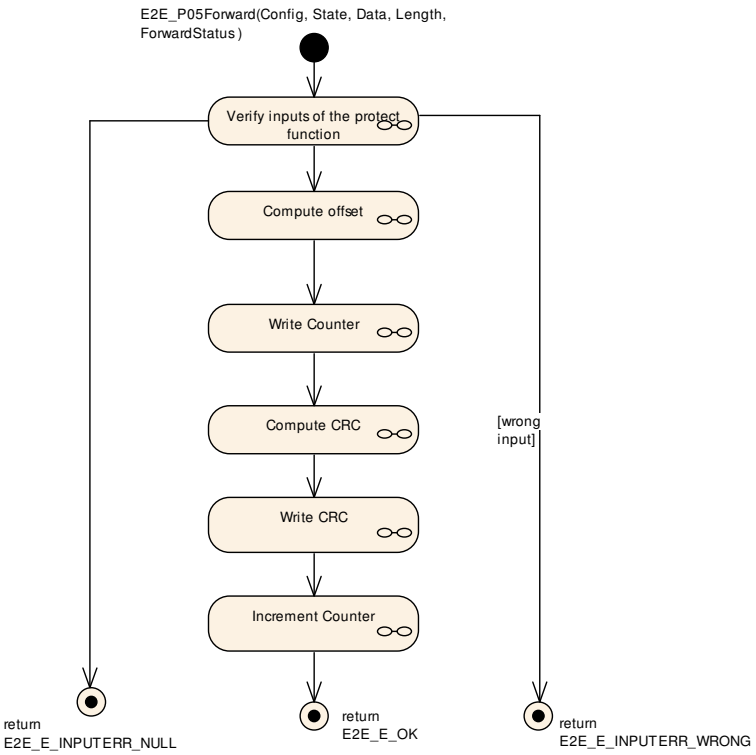

Figure 6.82: E2E Profile 5 Forward

Following steps are described in Section in Section 6.8.2.1

- ”Compute Offset” see [PRS_E2E_00469]
- ”Write CRC” see [PRS_E2E_00407]
- ”Increment Counter” see [PRS_E2E_00409]

[PRS_E2E_00619]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the forward function” in E2E_P05Forward() shall behave as shown in Figure 6.12.⌋

[PRS_E2E_00620]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P05Forward() shall behave as shown in Figure 6.13.⌋

[PRS_E2E_00621]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P05Forward() shall behave as shown in Figure 6.83.⌋

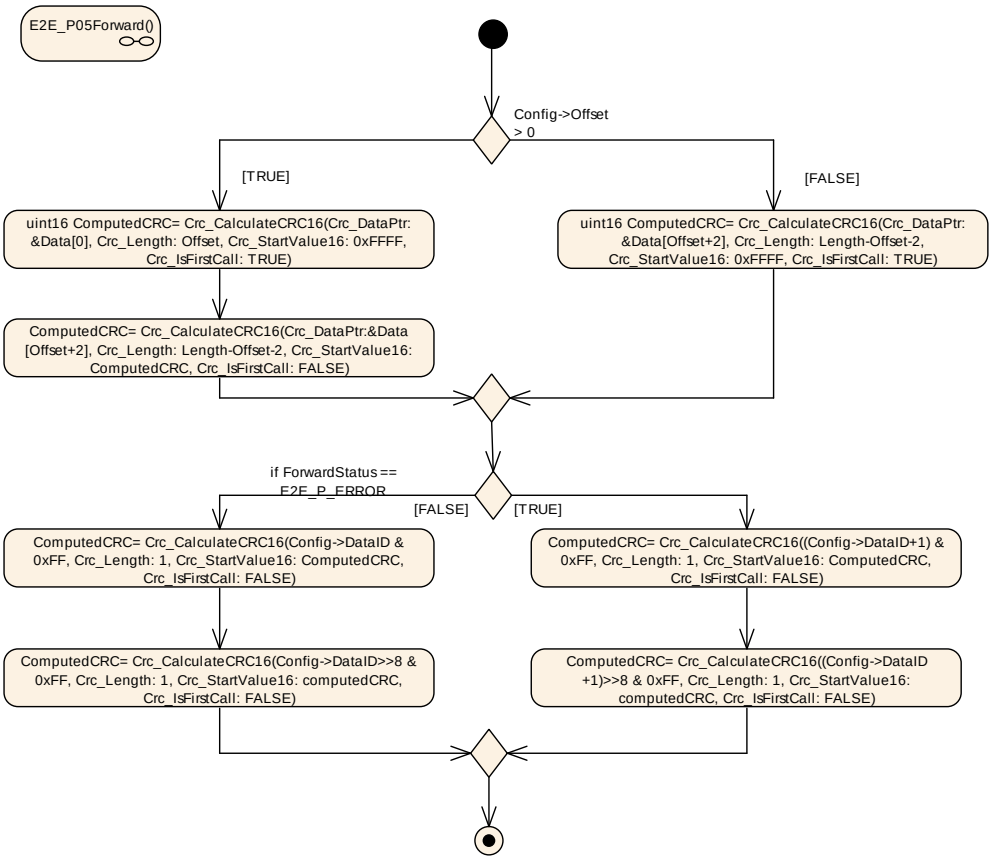

Figure 6.83: E2E Profile 5 Forward step ”Compute CRC”

6.8.3 Evaluation of the E2E-Header

6.8.3.1 E2E_P05Check

The function E2E_P05Check() performs the actions as specified by the following diagrams (see also Chapter 6.3).

[PRS_E2E_00411]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P05Check() shall have the overall behavior as shown in Figure 6.84.⌋

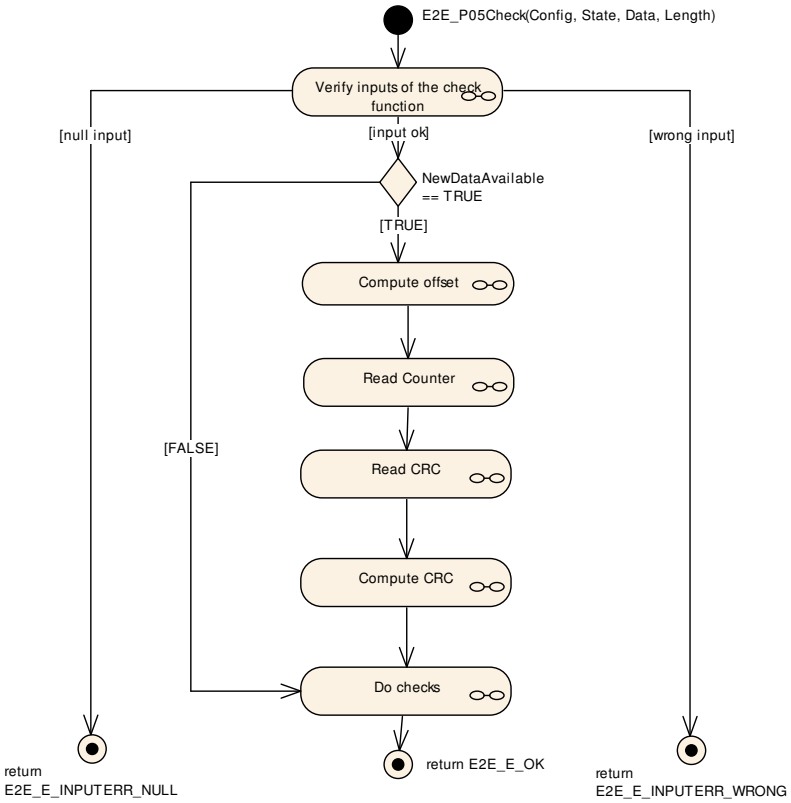

Figure 6.84: E2E Profile 5 Check

[PRS_E2E_00412]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the check function” in E2E_P05Check() shall behave as shown in Figure 6.16.⌋

[PRS_E2E_00413]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Counter” in E2E_P05Check() shall behave as shown in Figure 6.18.⌋

[PRS_E2E_00414]

Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_P05Check() shall behave as shown in Figure 6.20.⌋

[PRS_E2E_00416]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks’ in E2E_P05Check() shall behave as shown in Figure 6.21.⌋

6.8.4 Profile Data Types

6.8.4.1 Profile 5 Protect State Type

[PRS_E2E_00652]

Upstream requirements: RS_E2E_08528

⌈The E2E_P05Protect and E2E_P05Forward functions ’state’ shall have the members defined in [PRS_E2E_00878].⌋

[PRS_E2E_00878] E2E Profile 5 Protect State Type

Upstream requirements: RS_E2E_08528

⌈

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

⌋

6.8.4.2 Profile 5 Check Status Type

[PRS_E2E_00653]

Upstream requirements: RS_E2E_08528

⌈The E2E_P05Check functions ’State’ shall have the members defined in [PRS_E2E_00879].⌋

[PRS_E2E_00879] E2E Profile 5 Check Status Type

Upstream requirements: RS_E2E_08528

⌈

Member Name	State Type	Description

Counter	Unsigned Integer	Counter of the data in previous cycle.
Status	Enumeration	Result of the verification of the Data in this cycle, determined by the Check function.

⌋

6.8.4.3 Profile 5 Check Status Enumeration

[PRS_E2E_00591]

Upstream requirements: RS_E2E_08528

⌈The E2E_P05Check functions 'State->Status' enumeration type shall consist the following enumeration values (see [PRS_E2E_00880]).⌋

[PRS_E2E_00880] E2E Profile 5 Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_P05STATUS_OK	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
E2E_P05STATUS_NONEWDATA	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_P05STATUS_REPEATED.
E2E_P05STATUS_ERROR	Error	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
E2E_P05STATUS_REPEATED	Error	The checks of the Data in this cycle were successful, with the exception of the repetition.
E2E_P05STATUS_OKSOMELOST	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	Error	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta

⌋

6.8.4.4 Profile 5 Configuration Type

[PRS_E2E_00654]

Upstream requirements: RS_E2E_08528, RS_E2E_08537, RS_E2E_08539

⌈The E2E_P05Protect, E2E_P05Forward and E2E_P05Check functions 'Config' shall have the following members defined in [PRS_E2E_00881].⌋

[PRS_E2E_00881] E2E Profile 5 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537, RS_E2E_08539

⌈

Member Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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	Unsigned Integer	Length of Data, in bits. The value shall be <= 4096*8 (4kB) and shall be >= 3*8. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.8.5 E2E Profile 5 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P05ConfigType field	Value
DataID	0x1234
Offset	0x0000
DataLength	24
MaxDeltaCounter	1

Table 6.25: E2E Profile 5 protocol example configuration

E2E_P05ProtectStateType field	Value
Counter	0

Table 6.26: E2E Profile 5 example state initialization

Result data of E2E_P05Protect() with short data length (length 8 bytes, with 5 actual data bytes), offset = 0, counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x01c	0xca	0x00	0x00	0x00	0x00	0x00	0x00
Field	*CRC*		*Counter*	*Data*				

Table 6.27: E2E Profile 5 example short

Result data of E2E_P05Protect() with short data length (length 16 bytes, with 5 actual data bytes), offset = 64 (as with SOME/IP header use case), counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	8	9	10	11	12	13	14	15
Data	0x28	0x91	0x00	0x00	0x00	0x00	0x00	0x00
Field	*CRC*		*Counter*	*Data*				

Table 6.28: E2E Profile 5 example short with SOME/IP use case

6.9 Specification of E2E Profile 6

[PRS_E2E_00479]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈Profile 6 shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID (see [PRS_E2E_00882]).⌋

[PRS_E2E_00882] E2E Profile 6 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈

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

⌋

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

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

Table 6.29: Detectable communication faults using Profile 6

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P06LENGTH_POS	2
P06LENGTH_LEN	2
P06COUNTER_POS	4
P06COUNTER_LEN	1
P06DATAID_POS	0
P06DATAID_LEN	0
P06CRC_POS	0
P06CRC_LEN	2
P06CALCULATE_CRC	Crc_CalculateCRC16()

Table 6.30: Profile 6-specific data

6.9.1 Header layout

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

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

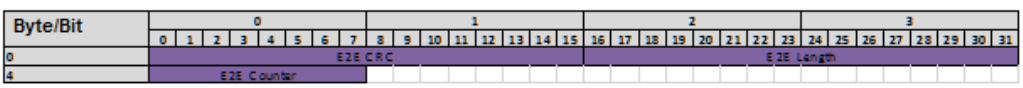

Figure 6.85: E2E Profile 6 header

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

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

6.9.1.1 Counter

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

[PRS_E2E_00417]

Upstream requirements: RS_E2E_08539

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

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

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

6.9.1.2 Data ID

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

[PRS_E2E_00419]

Upstream requirements: RS_E2E_08539

⌈In the E2E Profile 6, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation.⌋

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

[PRS_E2E_01442] Uniqueness of Data IDs

Replaces: PRS_E2E_UC_00464
Upstream requirements: RS_E2E_08539

⌈In the E2E profile 6, the Data IDs should be globally unique within the network of communicating system (made of several ECUs each sending different data).⌋

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

In case of usage of E2E Supervision for protecting messages (i.e. invocation from COM), the receiver COM expects at a reception only a specific message, which is checked by E2E Supervision using Data ID.

6.9.1.3 Length

In Profile 6 the length field is introduced to support variable-size length - the Data [] array storing the serialized data can potentially have a different length in each cycle. In Profile 6 there is an explicit transmission of the length. The Length includes user data + E2E Header (CRC + Counter + Length).

6.9.1.4 CRC

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

[PRS_E2E_00420]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E Profile 6 shall use the Crc_CalculateCRC16() function of the SWS CRC Supervision for calculating the CRC (Polynomial: 0x1021; Autosar notation).⌋

[PRS_E2E_00421]

Upstream requirements: RS_E2E_08539, RS_E2E_08531

⌈In E2E Profile 6, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended with the Data ID.⌋

6.9.2 Creation of E2E-Header

6.9.2.1 E2E_P06Protect

The function E2E_P06Protect() performs the steps as specified by the following diagrams (see also Chapter 6.3).

[PRS_E2E_00423]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P06Protect() shall have the overall behavior as shown in Figure 6.86.⌋

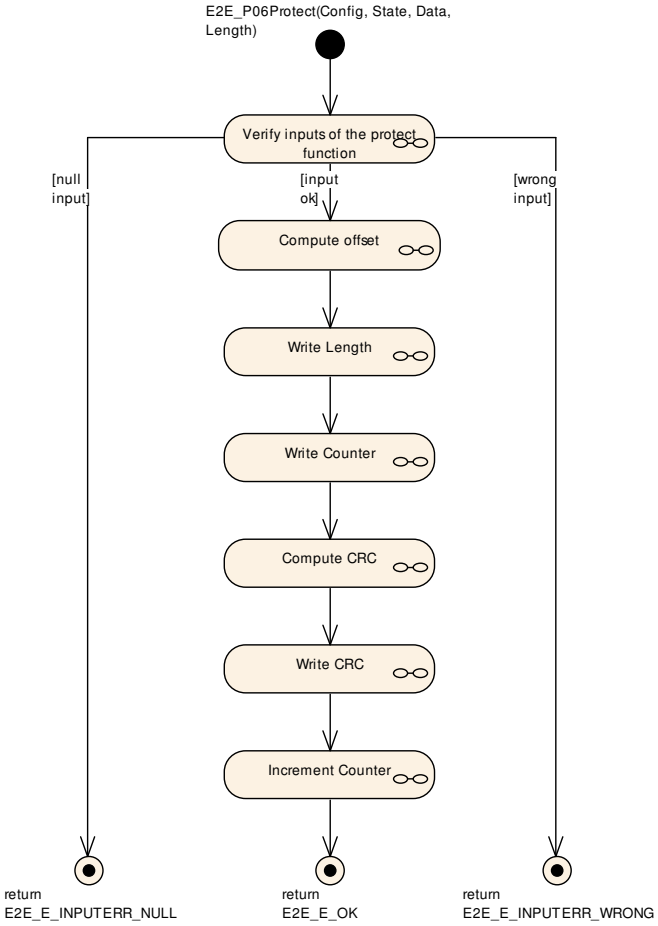

Figure 6.86: E2E Profile 6 Protect

[PRS_E2E_00424]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_P06Protect() shall behave as shown in Figure 6.3.⌋

[PRS_E2E_00470]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute offset” in E2E_P06Protect(), E2E_P06Forward() and E2E_P06Check() shall behave as shown in Figure 6.4.⌋

[PRS_E2E_00425]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Length” in E2E_P06Protect() and E2E_P06Forward() shall behave as shown in Figure 6.5.⌋

[PRS_E2E_00426]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P06Protect() shall behave as shown in Figure 6.6.⌋

[PRS_E2E_00427]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P06Protect() and E2E_P06Check() shall behave as shown in Figure 6.87.⌋

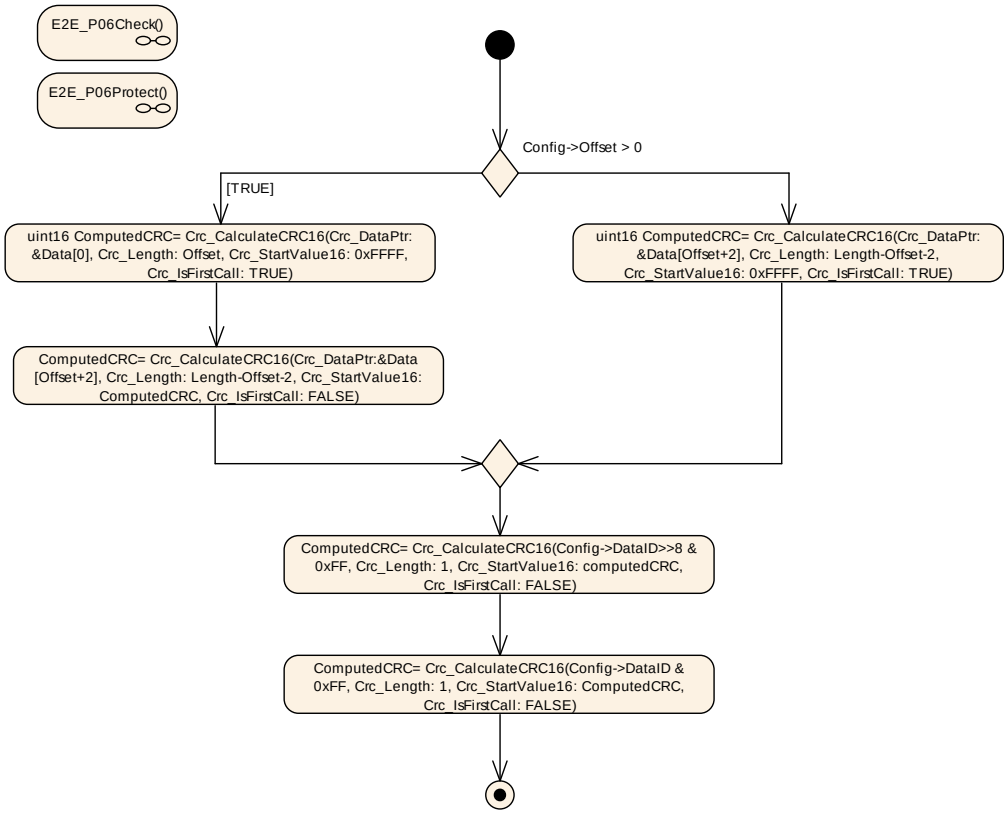

Figure 6.87: E2E Profile 6 Protect and Check step ”Compute CRC”

[PRS_E2E_00428]

Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC” in E2E_P06Protect() and E2E_P06Forward() shall behave as shown in Figure 6.9.⌋

[PRS_E2E_00429]

Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_P06Protect() and E2E_P06Forward() shall behave as shown in Figure 6.10.⌋

6.9.2.2 E2E_P06Forward

The E2E_P06Forward() function of E2E Profile 6 is called by a SW-C in order to protect its application data and forward an received E2E-Status for use cases like translation of signal based to service oriented communication. If the received E2E status equals

E2E_P_OK the behavior of the function shall be the same like E2E_P06Protect(). The function E2E_P06Forward() performs the steps as specified by the following diagrams (see also Chapter 6.3).

[PRS_E2E_00622]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P06Forward() shall have the overall behavior as shown in Figure 6.88.⌋

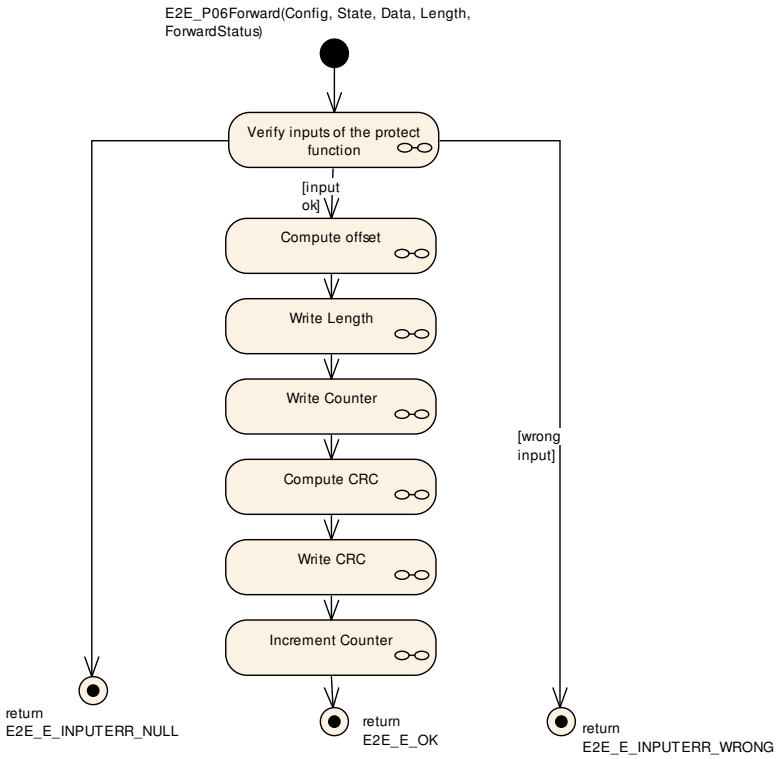

Figure 6.88: E2E Profile 6 Forward

Following steps are described in Section in Section 6.9.2.1

- ”Compute Offset” see [PRS_E2E_00470]
- ”Write Length” see [PRS_E2E_00425]
- ”Write CRC” see [PRS_E2E_00428]
- ”Increment Counter” see [PRS_E2E_00429]

[PRS_E2E_00623]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the forward function” in E2E_P06Forward() shall behave as shown in Figure 6.12.⌋

[PRS_E2E_00624]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P06Forward() shall behave as shown in Figure 6.13.⌋

[PRS_E2E_00625]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P06Forward() shall behave as shown in Figure 6.89.⌋

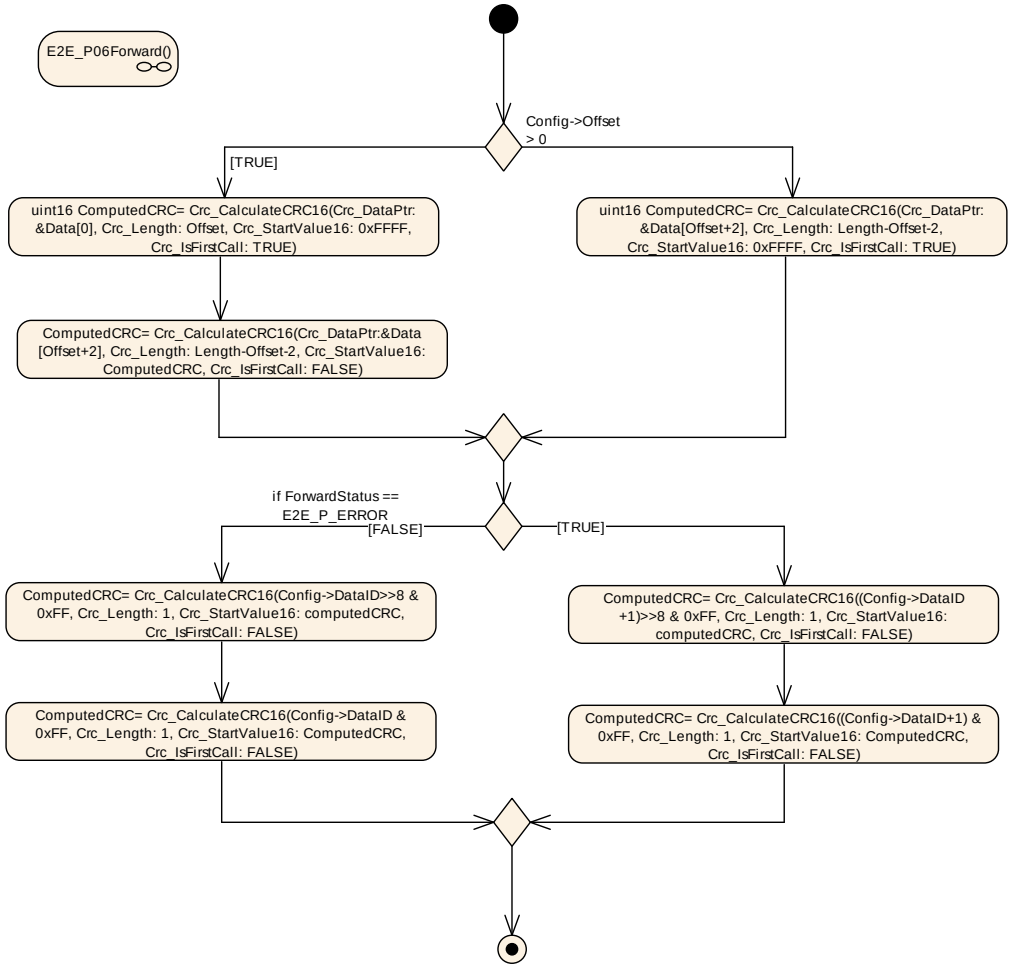

Figure 6.89: E2E Profile 6 Forward step ”Compute CRC”

6.9.3 Evaluation of E2E-Header

6.9.3.1 E2E_P06Check

The function E2E_P06Check() performs the actions as specified by the following diagrams (see also Chapter 6.3).

[PRS_E2E_00430]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P06Check() shall have the overall behavior as shown in Figure 6.90.⌋

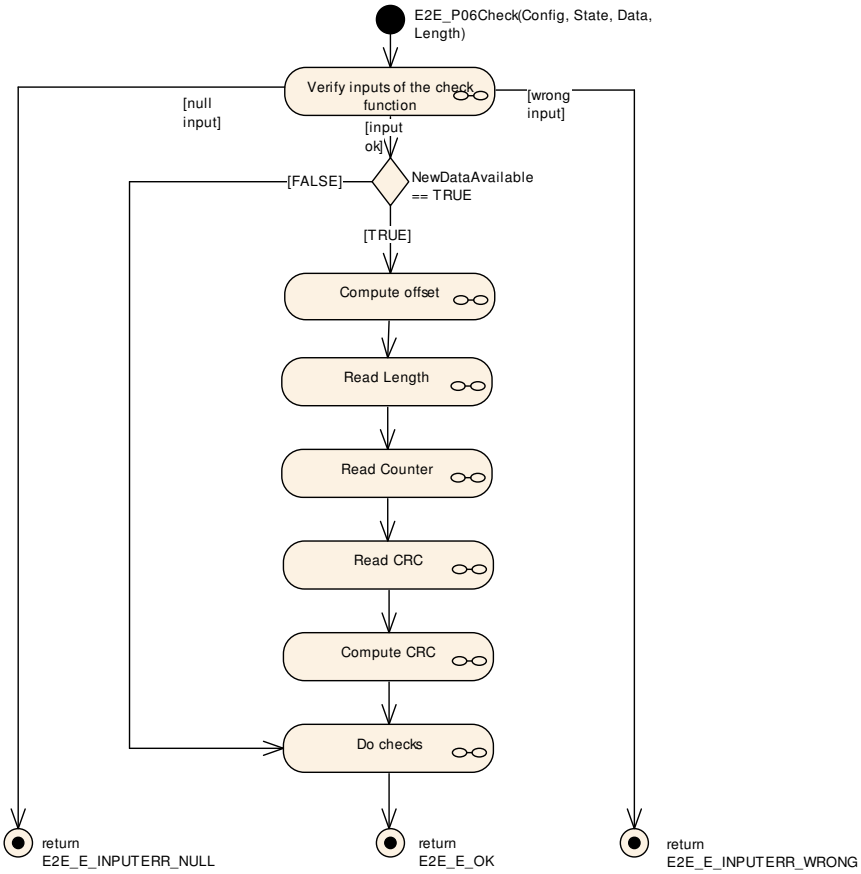

Figure 6.90: E2E Profile 6 Check

[PRS_E2E_00431]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify Inputs” in E2E_P06Check() shall behave as shown in Figure 6.16.⌋

[PRS_E2E_00432]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Length” in E2E_P06Check() shall behave as shown in Figure 6.17.⌋

[PRS_E2E_00433]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Counter” in E2E_P06Check() shall behave as shown in Figure 6.18.⌋

[PRS_E2E_00434]

Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_P06Check() shall behave as shown in Figure 6.20.⌋

[PRS_E2E_00436]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks” in E2E_P06Check() shall behave as shown in Figure 6.21.⌋

6.9.4 Profile Data Types

6.9.4.1 Profile 6 Protect State Type

[PRS_E2E_00655]

Upstream requirements: RS_E2E_08528

⌈The E2E_P06Protect and E2E_P06Forward functions ’state’ shall have the members defined in [PRS_E2E_00883].⌋

[PRS_E2E_00883] E2E Profile 6 Protect State Type

Upstream requirements: RS_E2E_08528

⌈

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

⌋

6.9.4.2 Profile 6 Check Status Type

[PRS_E2E_00656]

Upstream requirements: RS_E2E_08528

⌈The E2E_P06Check functions ’State’ shall have the members defined in [PRS_E2E_00884].⌋

[PRS_E2E_00884] E2E Profile 6 Check Status Type

Upstream requirements: RS_E2E_08528

⌈

Member Name	Type	Description
Counter	Unsigned Integer	Counter of the data in previous cycle.
Status	Enumeration	Result of the verification of the Data in this cycle, determined by the Check function.

⌋

6.9.4.3 Profile 6 Check Status Enumeration

[PRS_E2E_00592]

Upstream requirements: RS_E2E_08528

⌈The E2E_P06Check functions 'State->Status' enumeration type shall consist of the following enumeration values (see [PRS_E2E_00885]).⌋

[PRS_E2E_00885] E2E Profile 6 Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_P06STATUS_OK	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
E2E_P06STATUS_NONEWDATA	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_P06STATUS_REPEATED.
E2E_P06STATUS_ERROR	Error	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
E2E_P06STATUS_REPEATED	Error	The checks of the Data in this cycle were successful, with the exception of the repetition.
E2E_P06STATUS_OKSOMELOST	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	Error	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta

⌋

6.9.4.4 Profile 6 Configuration Type

[PRS_E2E_00657]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P06Protect, E2E_P06Forward and E2E_P06Check functions 'Config' shall have the following members defined in [PRS_E2E_00886].⌋

[PRS_E2E_00886] E2E Profile 6 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	State Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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 Length (16 bit) is written to Byte 1, the low Byte is written to Byte 2. This may be considered similar to E2E_P06STATUS_REPEATED.
MinDataLength	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be <= MaxDataLength and shall be >= 5*8. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 4096*8 (4KB) and it shall be >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	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.

⌋

6.9.5 E2E Profile 6 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P06ConfigType field	Value
DataID	0x1234
Offset	0x0000
MinDataLength	40
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.31: E2E Profile 6 protocol example configuration

E2E_P06ProtectStateType field	Value
Counter	0

Table 6.32: E2E Profile 6 example state initialization

Result data of E2E_P06Protect() with short data length (length 8 bytes, with 3 actual data bytes), offset = 0, counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0xb1	0x55	0x00	0x08	0x00	0x00	0x00	0x00
Field	*CRC*		*Length*		*Counter*	*Data*		

Table 6.33: E2E Profile 6 example short

Result data of E2E_P06Protect() with short data length (length 16 bytes, with 3 actual data bytes), offset = 64 (as with SOME/IP header use case), counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	8	9	10	11	12	13	14	15
Data	0x4e	0xb7	0x00	0x10	0x00	0x00	0x00	0x00
Field	*CRC*		*Length*		*Counter*	*Data*		

Table 6.34: E2E Profile 6 example short with SOME/IP use case

6.10 Specification of E2E Profile 7

[PRS_E2E_00480]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548

⌈Profile 7 shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID (see [PRS_E2E_00904]).⌋

[PRS_E2E_00904] E2E Profile 7 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548

⌈

Control field	Description
Length	32 bits, to support dynamic-size data.
Counter	32 bits.

CRC	64 bits, polynomial in normal form 0x42F0E1EBA9EA3693, provided by CRC library. Note: This CRC polynomial is also known as “CRC-64 (ECMA)”.
Data ID	32 bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P07LENGTH_POS	8
P07LENGTH_LEN	4
P07COUNTER_POS	12
P07COUNTER_LEN	4
P07DATAID_POS	16
P07DATAID_LEN	4
P07CRC_POS	0
P07CRC_LEN	8
P07CALCULATE_CRC	Crc_CalculateCRC64()

Table 6.35: Profile 7-specific data

For behavior and flowcharts of E2E Profile 07 see Chapter 6.3.

6.10.1 Header layout

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

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

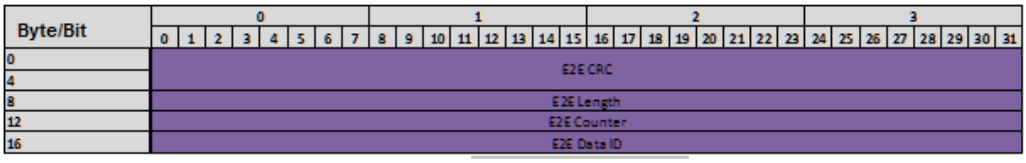

Figure 6.91: Profile 7 Header

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

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

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

6.10.2 Profile 7 Configuration Type

[PRS_E2E_00660]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P07Protect, E2E_P07Forward and E2E_P07Check functions 'Config' shall have the following members defined in [PRS_E2E_00887].⌋

[PRS_E2E_00887] E2E Profile 7 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be >= 20*8 and <= MaxDataLength. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be >= MinDataLength and it should be <= 4194304*8. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.10.3 E2E Profile 7 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P07ConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000
MinDataLength	160
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.36: E2E Profile 7 protocol example configuration

E2E_P07ProtectStateType field	Value
Counter	0

Table 6.37: E2E Profile 7 example state initialization

Result data of E2E_P07Protect() with short data length (length 24 bytes, means 4 actual data bytes), offset = 0, counter = 0:

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data	0x1f	0xb2	0xe7	0x37	0xfc	0xed	0xbc	0xd9
Field	*CRC*							
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0x00	0x00	0x00	0x18	0x00	0x00	0x00	0x00
Field	*Length*				*Counter*			
Byte	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Data	0x0a	0x0b	0x0c	0x0d	0x00	0x00	0x00	0x00
Field	*DataID*				*Data*			

Table 6.38: E2E Profile 7 example short

Result data of E2E_P07Protect() with short data length (length 32, means 4 actual data bytes), offset = 64 (as with SOME/IP header use case), counter = 0:

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0x17	0xf7	0xc8	0x17	0x32	0x38	0x65	0xa8
Field	*CRC*							
Byte	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Data	0x00	0x00	0x00	0x20	0x00	0x00	0x00	0x00
Field	*Length*				*Counter*			
Byte	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Data	0x0a	0x0b	0x0c	0x0d	0x00	0x00	0x00	0x00
Field	*DataID*				*Data*			

Table 6.39: E2E Profile 7 example short with SOME/IP use case

6.11 Specification of E2E Profile 8

[PRS_E2E_00736]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08549

⌈Profile 8 shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID (see [PRS_E2E_00901]).⌋

[PRS_E2E_00901] E2E Profile 8 mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08549

⌈

Control field	Description
Length	32 bits, to support dynamic-size data.
Counter	32 bits.
CRC	32 bits, polynomial in normal form 0xF4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC polynomials used by FlexRay, CAN and LIN and TCPIP.
Data ID	32 bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P08LENGTH_POS	4
P08LENGTH_LEN	4
P08COUNTER_POS	8
P08COUNTER_LEN	4
P08DATAID_POS	12
P08DATAID_LEN	4
P08CRC_POS	0
P08CRC_LEN	4
P08CALCULATE_CRC	Crc_CalculateCRC32P4()

Table 6.40: Profile 8-specific data

For behavior and flowcharts of E2E Profile 08 see Chapter 6.3.

6.11.1 Header layout

In the E2E Profile 8, the user data layout (of the data to be protected) is not constrained by E2E Profile 8 - there is only a requirement that the length of data to be protected is multiple of 1 byte.

The header of the E2E Profile 8 has one fixed layout, as follows:

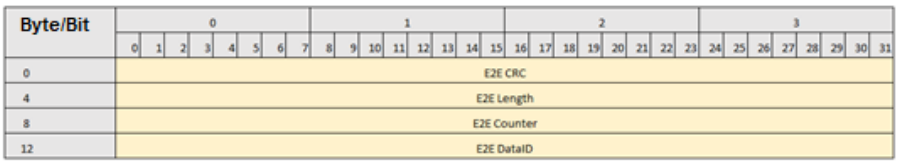

Figure 6.92: Profile 8 Header

The E2E header fields (e.g. E2E Counter) are encoded as:

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

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

6.11.2 Profile 8 Configuration Type

[PRS_E2E_00706]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P08Protect, E2E_P08Forward and E2E_P08Check functions 'Config' shall have the following members defined in [PRS_E2E_00902].⌋

[PRS_E2E_00902] E2E Profile 8 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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 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	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be >= 16*8 and <= MaxDataLength. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 536870912*8 and >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.11.3 E2E Profile 8 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P08ConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000 0000
MinDataLength	128
MaxDataLength	4294967288
MaxDeltaCounter	1

Table 6.41: E2E Profile 8 protocol example configuration

E2E_P08ProtectStateType field	Value
Counter	0

Table 6.42: E2E Profile 8 example state initialization

Result data of E2E_P08Protect() with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0:

Byte	1	2	3	4	5	6	7	8
Data	0x41	0x49	0x4e	0x52	0x00	0x00	0x00	0x14
Field	*CRC32*				*Length*			
Byte	9	10	11	12	13	14	15	16
Data	0x00	0x00	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Counter*				*DataID*			
Byte	17	18	19	20				
Data	0x00	0x00	0x00	0x00				
Field	*Data*							

Table 6.43: E2E Profile 8 example short

Result data of E2E_P08Protect() with minimum data length (4 data bytes), offset = 64 (as with SOME/IP header use case), datalength = 28, counter = 0:

Byte	1	2	3	4	5	6	7	8
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	9	10	11	12	13	14	15	16
Data	0xe8	0x91	0xe5	0xa8	0x00	0x00	0x00	0x1c
Field	*CRC32*				*Length*			
Byte	17	18	19	20	21	22	23	24
Data	0x00	0x00	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Counter*				*DataID*			

▽

△

Byte	25	26	27	28	
Data	0x00	0x00	0x00	0x00	
Field	*Data*				

Table 6.44: E2E Profile 8 example short with SOME/IP use case

6.12 Specification of E2E Profile 11

Profile 11 is bus-compatible to profile 1, but provides "new" profile behavior similar to profiles 4 to 7 on receiver side. Moreover, the following legacy DataIDModes are by now obsolete and omitted: E2E P11 DATAID LOW, E2E P11 DATAID ALT.

[PRS_E2E_00503]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈Profile 11 shall provide the following control fields, transmitted at runtime together with the protected data: Counter, CRC, Data ID (see [PRS_E2E_00888]).⌋

[PRS_E2E_00888] E2E Profile 11 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈

Control field	Description
Counter	4 bits. (explicitly sent)
CRC	8 bits, CRC-8-SAE J1850, provided by CRC library. (explicitly sent)
Data ID	16 bits or 12 bit, unique system-wide. (either implicitly sent (16 bits) or partly explicitly sent (12 bits; 4 bits explicitly and 8 bits implicitly sent))

⌋

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

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC

Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of receivers and the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 6.45: Detectable communication faults using Profile 11

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

6.12.1 Header Layout

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

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

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

Figure 6.93: E2E Profile 11 Variant C

The figure above shows Profile 11 variant 11C where the configuration is given as: The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

1. CRCOffset = 0
2. CounterOffset = 8 by Flexray/CAN bus.
3. DataIDNibbleOffset = 12

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

[PRS_E2E_00540]

Upstream requirements: RS_E2E_08528

⌈The E2E Profile variant 11A is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P11DataIDMode = E2E_P11_DATAID_BOTH

4. SignalIPdu.unusedBitPattern = 0xFF.

⌋

Below is an example compliant to 11A:

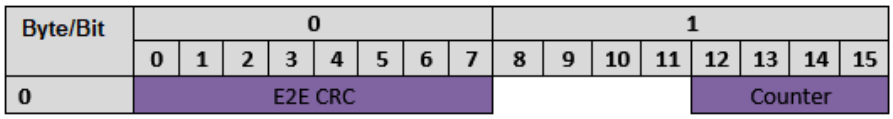

Figure 6.94: E2E Profile 11 Variant A

[PRS_E2E_00541]

Upstream requirements: RS_E2E_08528

⌈The E2E Profile variant 11C is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1st byte (i.e. starts with bit offset 12)
4. E2E_P11DataIDMode = E2E_P11_DATAID_NIBBLE
5. SignalIPdu.unusedBitPattern = 0xFF

⌋

E2E Profile variants 11A and 11C relate to the recommended Configuration of E2E Profile 11 configuration settings 11A and 11C in system template (system template is more specific).

For comparability to the figures of profile 1 the bit order is given. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

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

6.12.1.1 Counter

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

[PRS_E2E_00504]

Upstream requirements: RS_E2E_08539

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

Note that the counter value 0x0F is reserved as a special invalid value, and shall never be used by the E2E profile 11.

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

6.12.1.2 Data ID

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

[PRS_E2E_00583]

Upstream requirements: RS_E2E_08539

⌈The following two Data ID modes shall be supported:

1. E2E_P11_DATAID_BOTH: both bytes of the 16 bit Data ID are used in the CRC calculation: first the low byte and then the high byte.
2. E2E_P11_DATAID_NIBBLE:

   the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,

   the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.

   the low byte is not transmitted, but it is included in the CRC computation as start value.

⌋

[PRS_E2E_00507]

Upstream requirements: RS_E2E_08539

⌈In the E2E profile 11, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data).⌋

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

In case of usage of E2E Supervision for protecting messages (i.e. invocation from COM), the receiver COM expects at a reception only a specific message, which is checked by E2E Supervision using Data ID.

6.12.1.3 Length

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

6.12.1.4 CRC

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

[PRS_E2E_00508]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E Profile 11 shall use the Crc_CalculateCRC8 function of the SWS CRC Supervision for calculating the CRC (CRC-8-SAE J1850).⌋

[PRS_E2E_00505]

Upstream requirements: RS_E2E_08539

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

[PRS_E2E_00506]

Upstream requirements: RS_E2E_08539

⌈In E2E Profile 11 with DataIDMode set to E2E_P11_DATAID_NIBBLE, the lower nibble of the high byte of the DataID shall be placed in the transmitted data at bit position DataIDNibbleOffset, and the CRC calculation shall be done by first calculating over the low byte of the Data ID, then a 0-byte, and then the user data.⌋

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

6.12.2 Creation of the E2E-Header

6.12.2.1 E2E_P11Protect

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

[PRS_E2E_00509]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P11Protect() shall have the overall behavior as shown in Figure 6.95.⌋

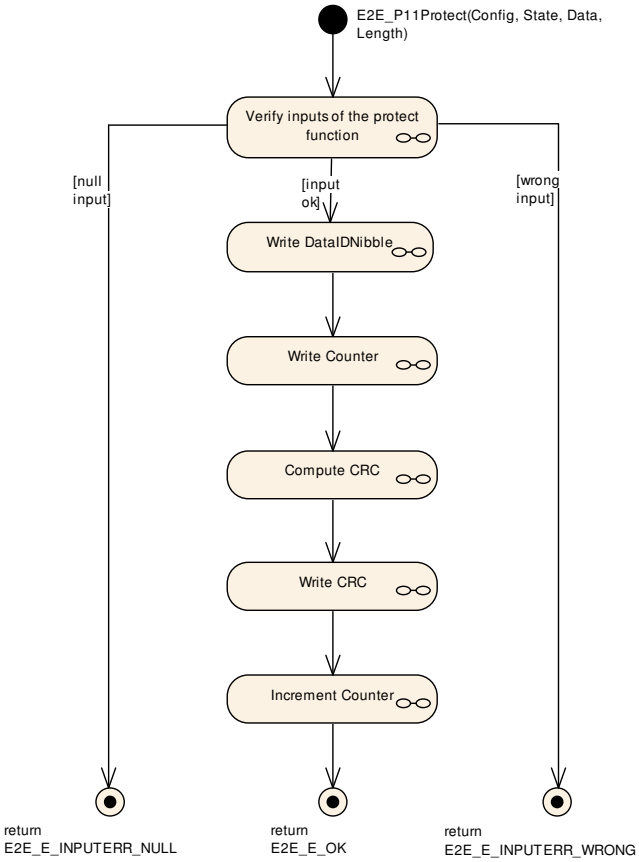

Figure 6.95: E2E Profile 11 Protect

[PRS_E2E_00510]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_P11Protect() shall behave as shown in Figure 6.96.⌋

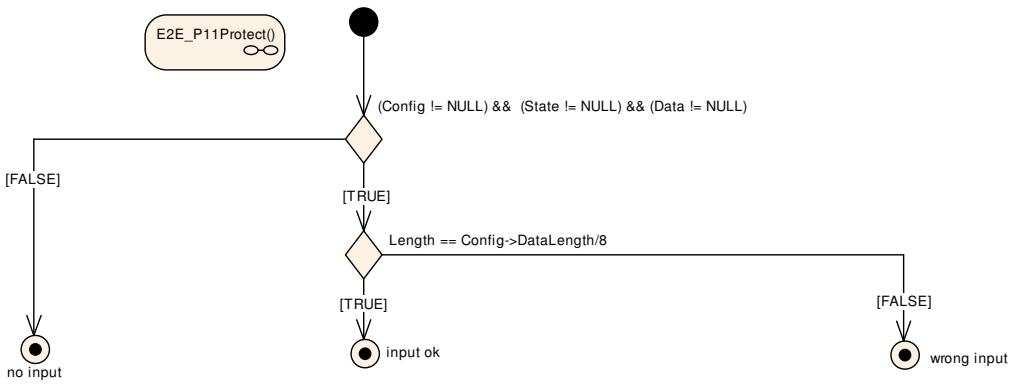

Figure 6.96: E2E Profile 11 Protect step ”Verify inputs of the protect function”

[PRS_E2E_00511]

Upstream requirements: RS_E2E_08539

⌈The step „Write DataIDNibble" in E2E_P11Protect() shall behave as shown in Figure 6.97.⌋

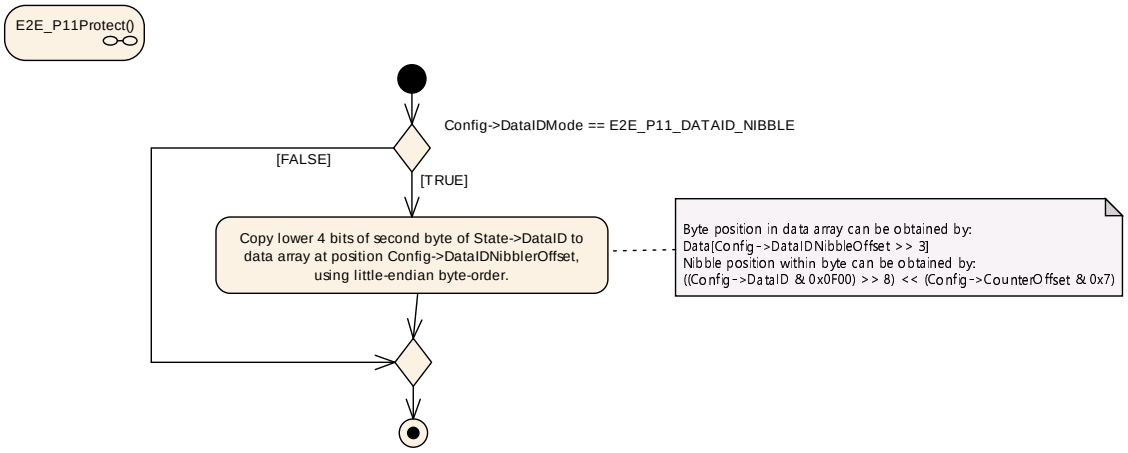

Figure 6.97: E2E Profile 11 Protect step "Write DataIDNibble"

[PRS_E2E_00512]

Upstream requirements: RS_E2E_08539

⌈The step "Write Counter" in E2E_P11Protect() shall behave as shown in Figure 6.98.⌋

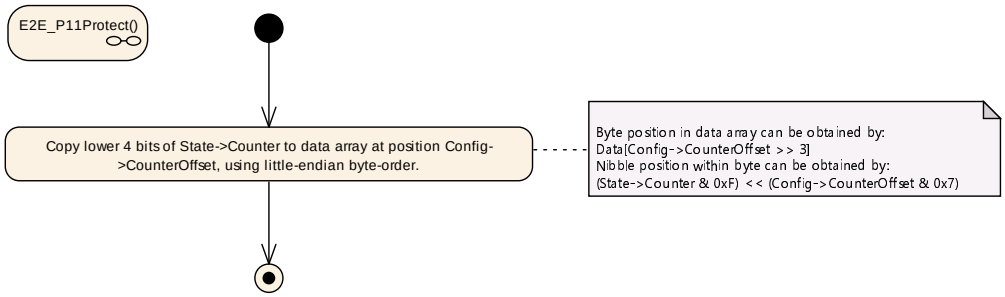

Figure 6.98: E2E Profile 11 Protect step "Write Counter"

[PRS_E2E_00513]

Upstream requirements: RS_E2E_08539

⌈The step "Compute CRC" in E2E_P11Protect() and in E2E_P11Check shall behave as shown in Figure 6.99.⌋

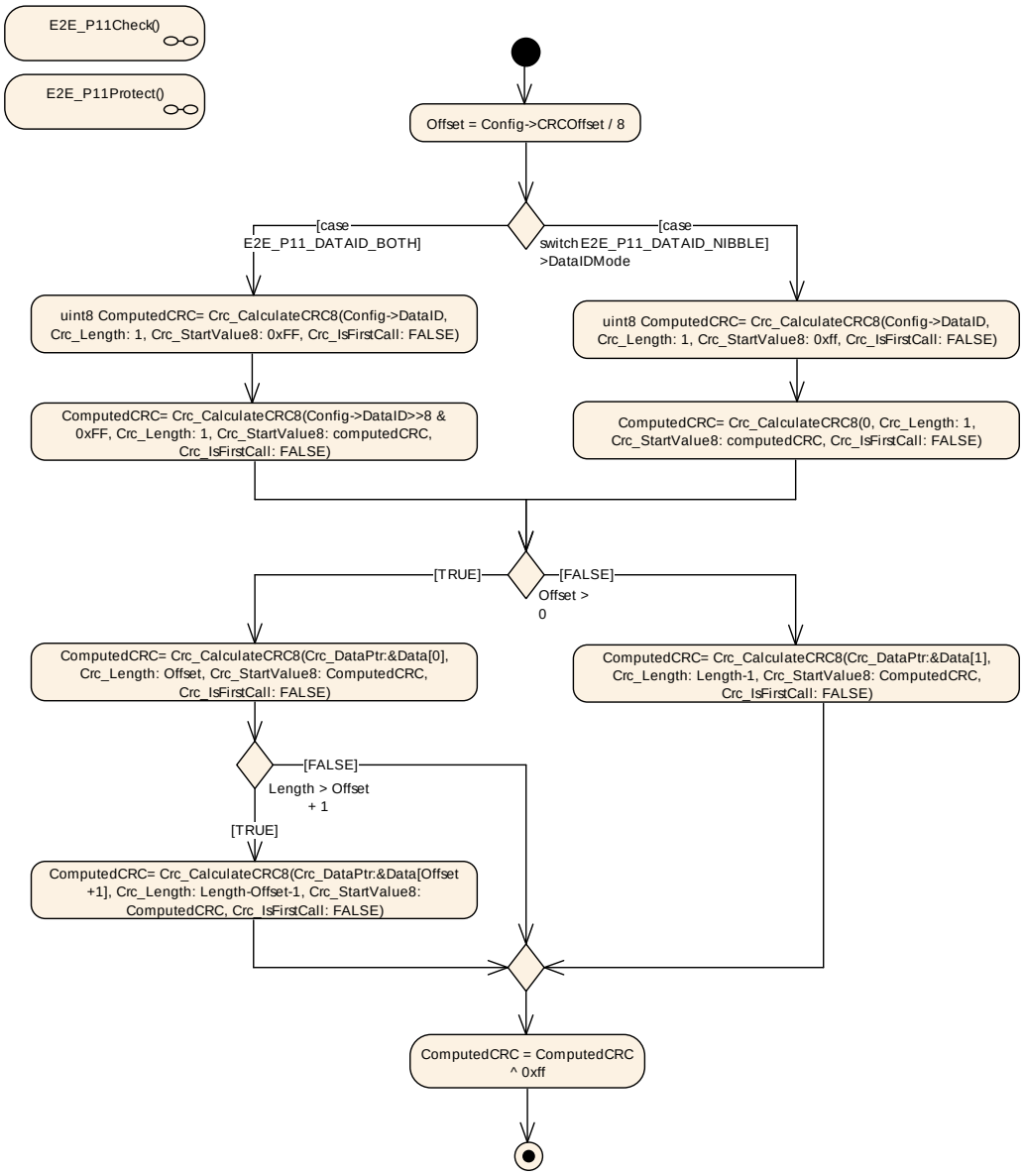

Figure 6.99: E2E Profile 11 Protect and Check step ”Compute CRC”

[PRS_E2E_00514]

Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC” in E2E_P11Protect() and E2E_P11Forward() shall behave as shown in Figure 6.100.⌋

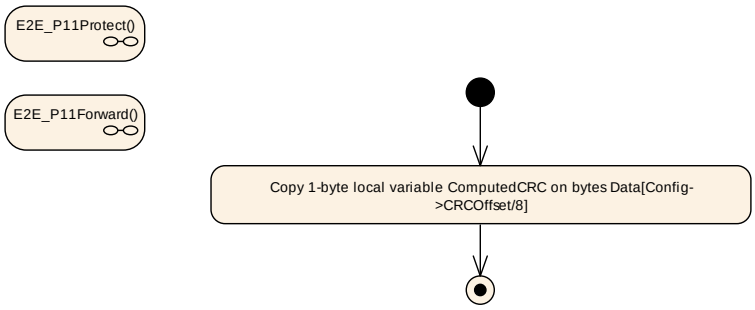

Figure 6.100: E2E Profile 11 Protect and Forward step ”Write CRC”

[PRS_E2E_00515]

Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_P11Protect() and E2E_P11Forward() shall behave as shown in Figure 6.101.⌋

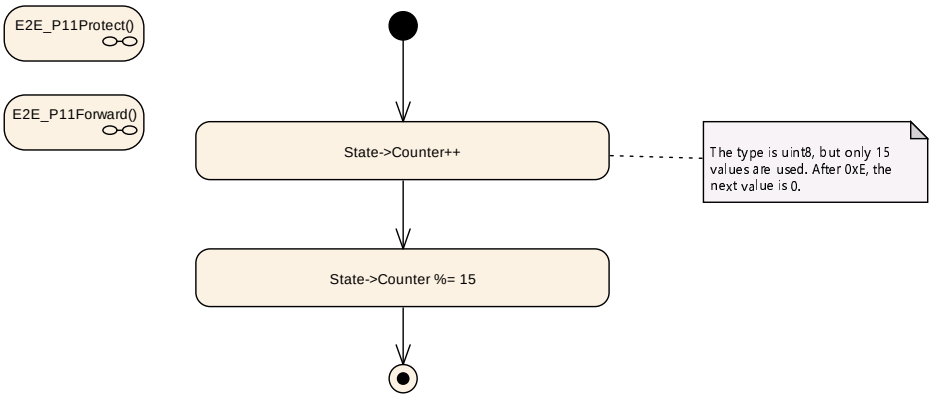

Figure 6.101: E2E Profile 11 Protect and Forward step ”Increment Counter”

6.12.2.2 E2E_P11Forward

The E2E_P11Forward() function of E2E Profile 11 is called by a SW-C in order to protect its application data and forward an received E2E-Status for use cases like translation of signal based to service oriented communication. If the received E2E status equals E2E_P_OK the behavior of the function shall be the same like E2E_P11Protect(). The function E2E_P11Forward() performs the steps as specified by the following five diagrams in this section.

[PRS_E2E_00630]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P11Forward() shall have the overall behavior as shown in Figure 6.102.⌋

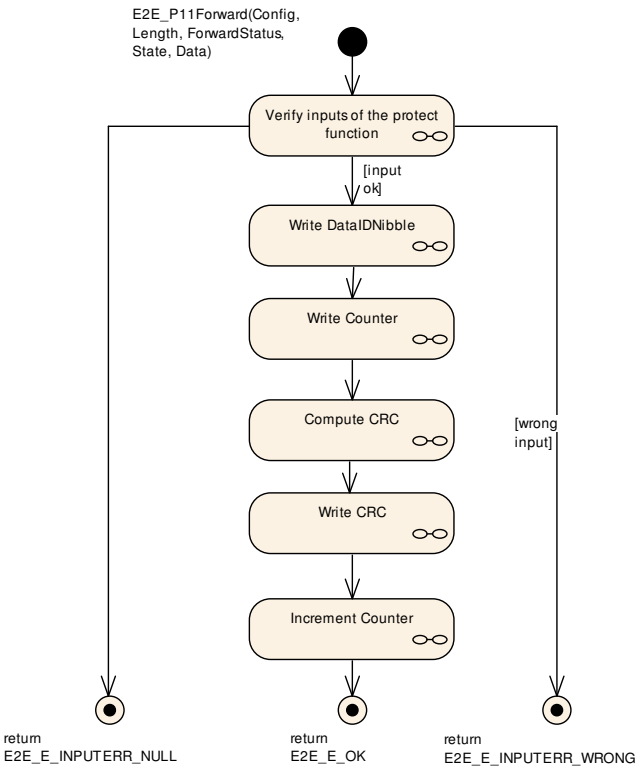

Figure 6.102: E2E Profile 11 Forward

Following steps are described in Section in Section 6.12.2.1

- ”Write CRC” see [PRS_E2E_00514]
- ”Increment Counter” see [PRS_E2E_00515]

[PRS_E2E_00631]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the forward function” in E2E_P11Forward() shall behave as shown in Figure 6.103.⌋

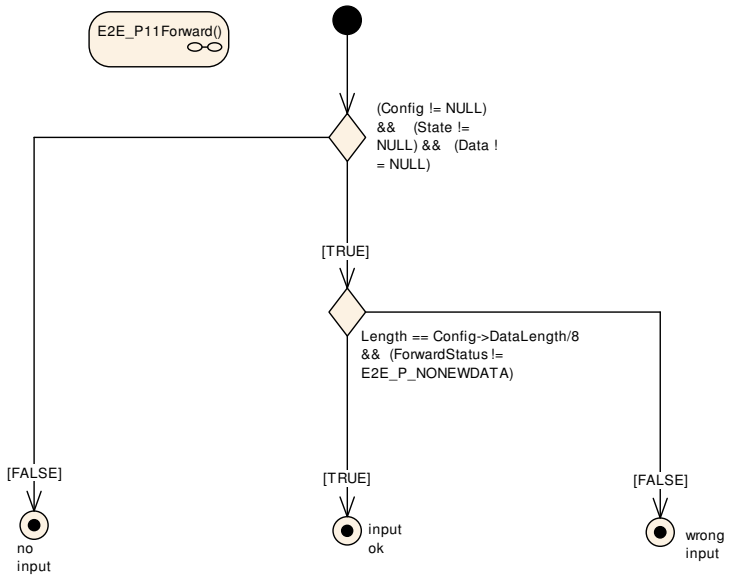

Figure 6.103: E2E Profile 11 Forward step ”Verify inputs of the forward function”

[PRS_E2E_00632]

Upstream requirements: RS_E2E_08539

⌈The step „Write DataIDNibble” in E2E_P11Forward() shall behave as shown in Figure 6.104.⌋

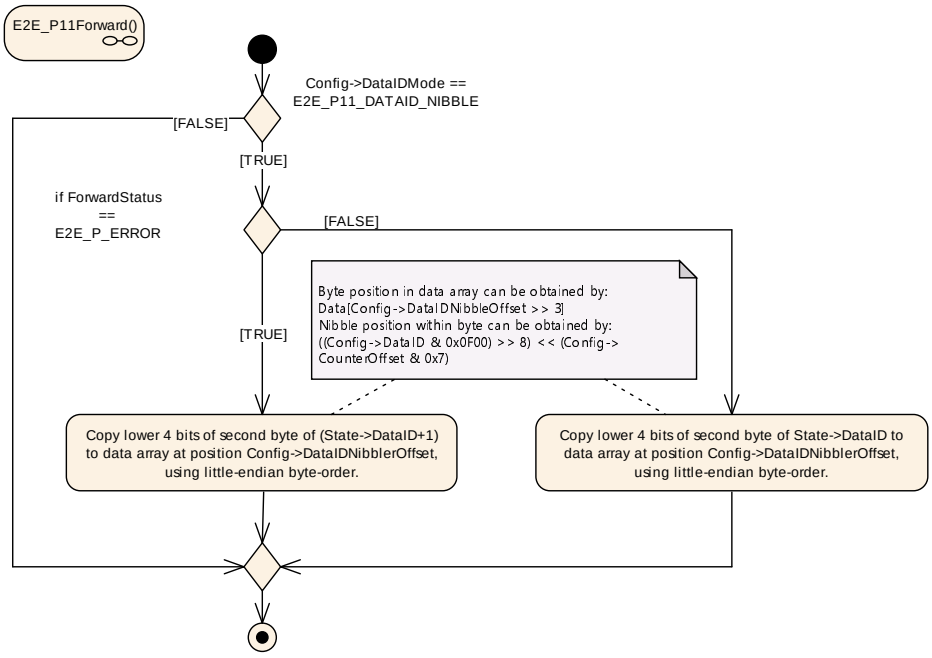

Figure 6.104: E2E Profile 11 Forward step ”Write DataIDNibble”

[PRS_E2E_00633]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P11Forward() shall behave as shown in Figure 6.105.⌋

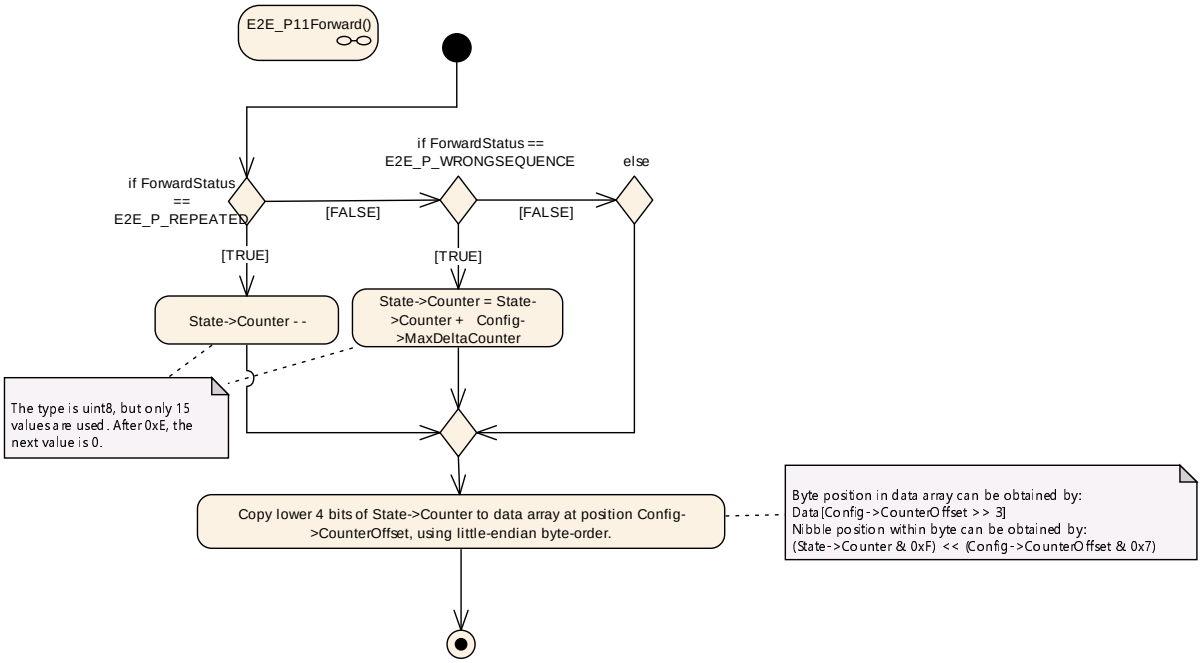

Figure 6.105: E2E Profile 11 Forward step ”Write Counter”

[PRS_E2E_00634]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P11Forward() shall behave as shown in Figure 6.106.⌋

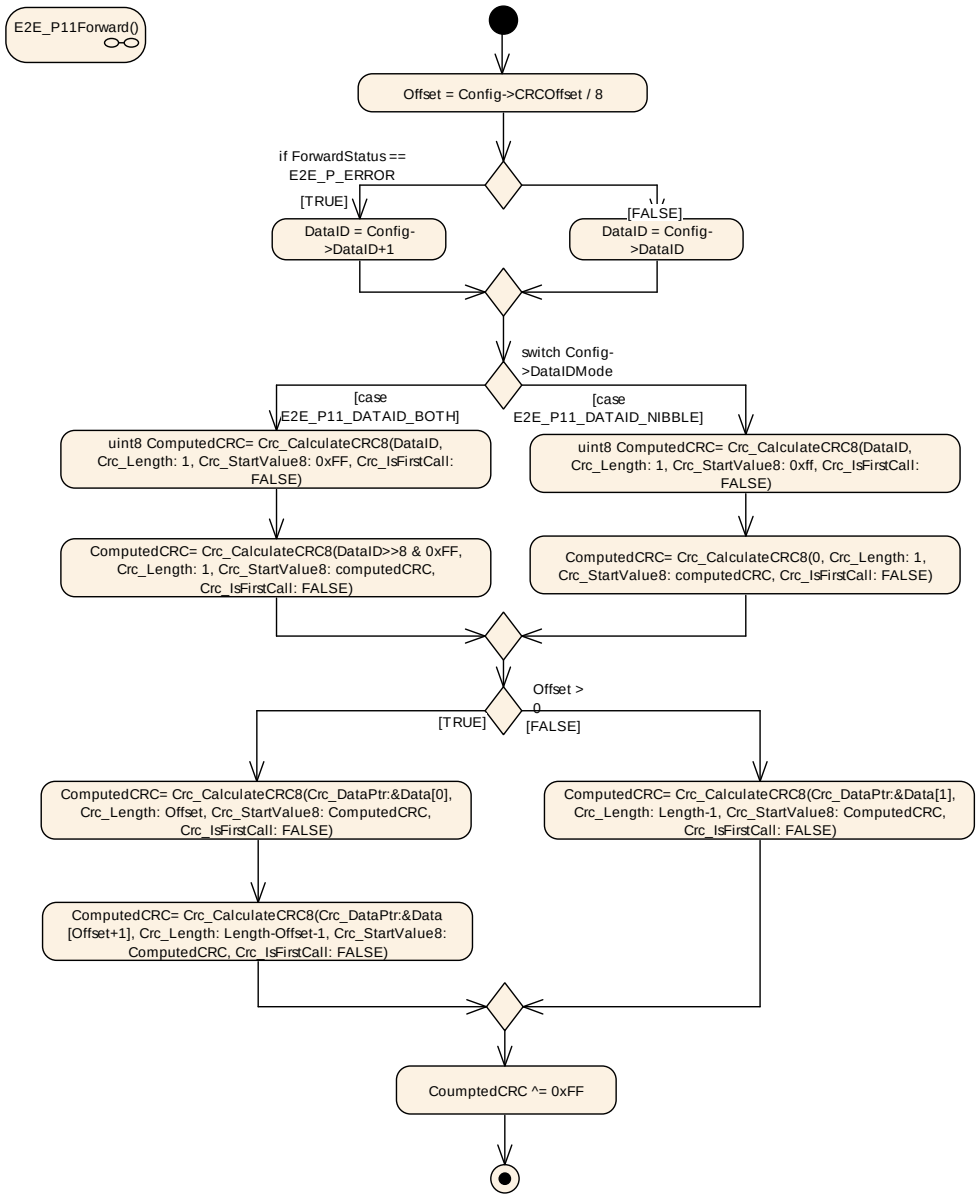

Figure 6.106: E2E Profile 11 Forward step ”Compute CRC”

6.12.3 E2E_P11Check

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

[PRS_E2E_00516]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P11Check() shall have the overall behavior as shown in Figure 6.107.⌋

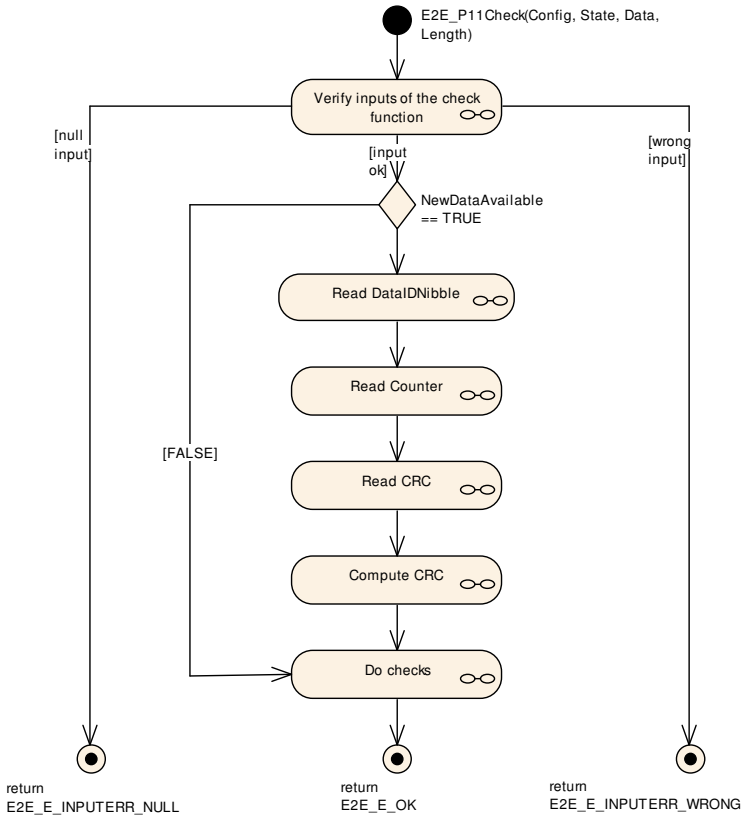

Figure 6.107: E2E Profile 11 Check

[PRS_E2E_00517]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the check function” in E2E_P11Check() shall behave as shown in Figure 6.108.⌋

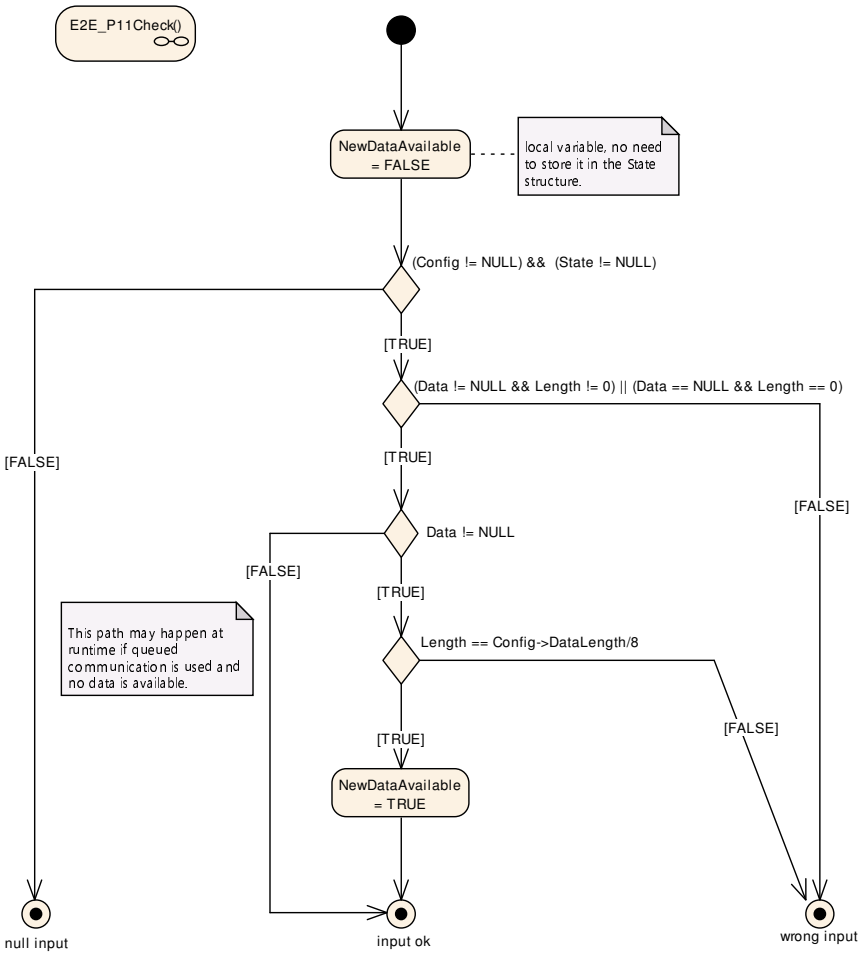

Figure 6.108: E2E Profile 11 Check step ”Verify inputs of the check function”

[PRS_E2E_00582]

Upstream requirements: RS_E2E_08539

⌈The step ”Read DataIDNibble” in E2E_P11Check() shall behave as shown in Figure 6.109.⌋

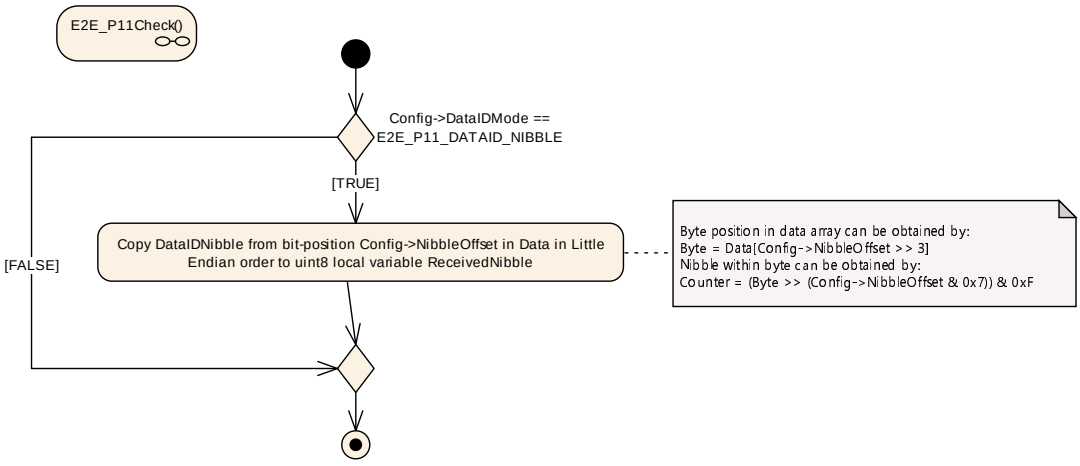

Figure 6.109: E2E Profile 11 Check step ”Read DataIDNibble”

[PRS_E2E_00518]

Upstream requirements: RS_E2E_08539

⌈The step "Read Counter" in E2E_P11Check() shall behave as shown in Figure 6.110.⌋

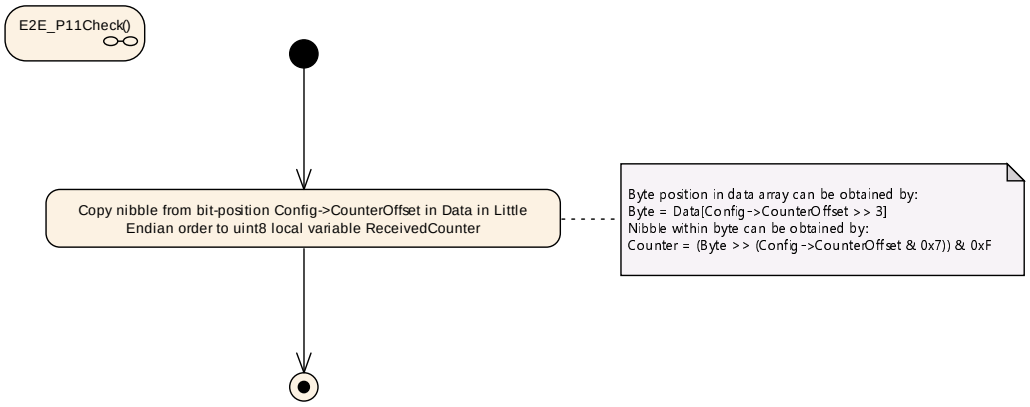

Figure 6.110: E2E Profile 11 Check step ”Read Counter”

[PRS_E2E_00519]

Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_P11Check() shall behave as shown in Figure 6.111.⌋

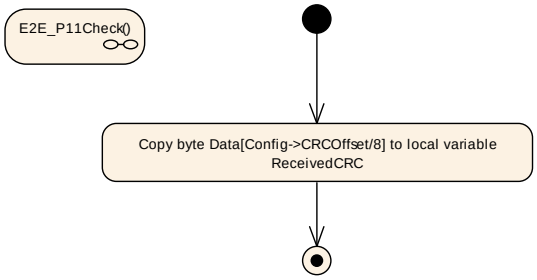

Figure 6.111: E2E Profile 11 Check step ”Read CRC”

[PRS_E2E_00521]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks’ in E2E_P11Check() shall behave as shown in Figure 6.112.⌋

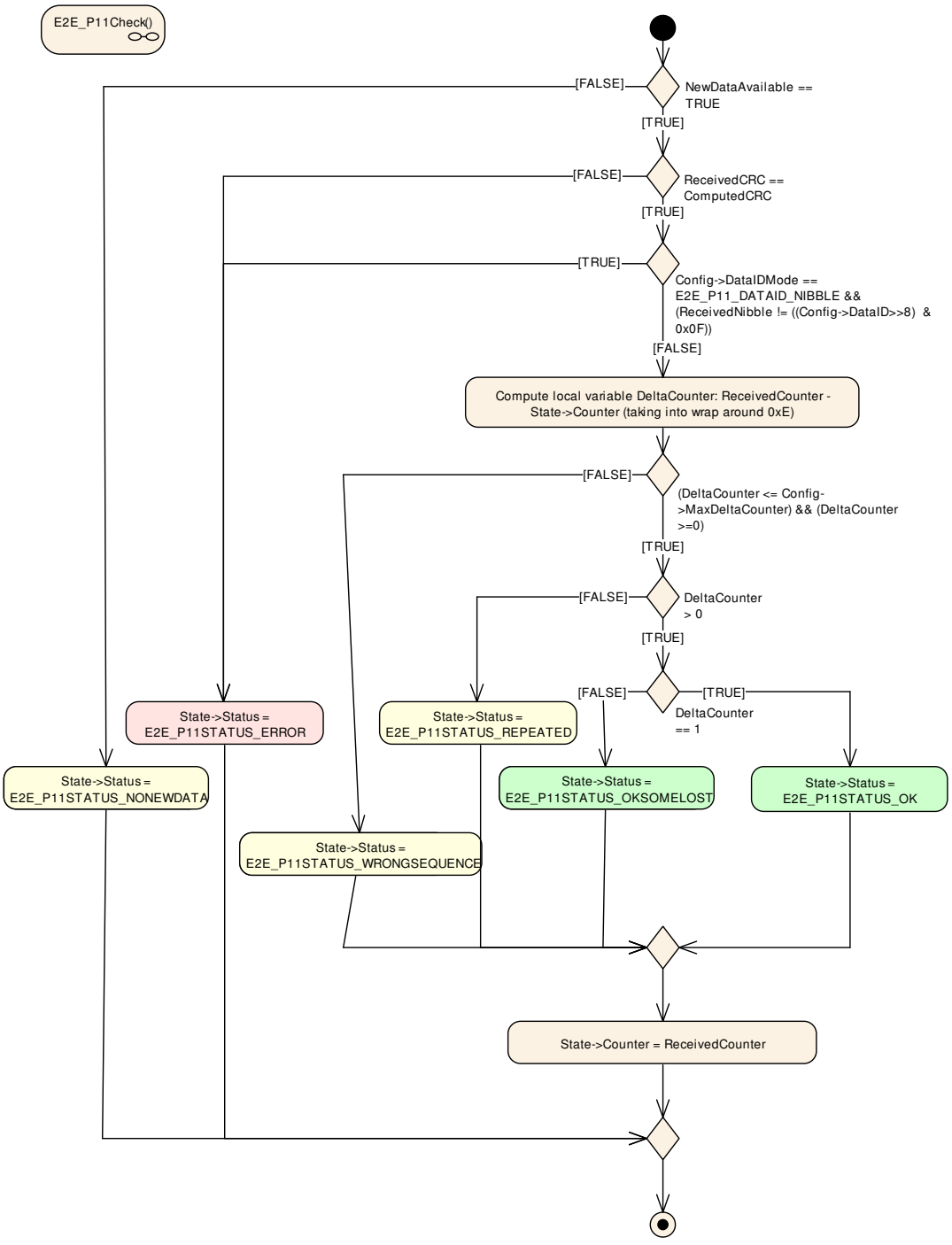

Figure 6.112: E2E Profile 11 Check step ”Do Checks”

6.12.4 Profile 11 Data Types

6.12.4.1 Profile 11 Protect State Type

[PRS_E2E_00661]

Upstream requirements: RS_E2E_08528

⌈The E2E_P11Protect and E2E_P11Forward functions 'State' shall have the members defined in [PRS_E2E_00889].⌋

[PRS_E2E_00889] E2E Profile 11 Protect State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
Counter	Unsigned Integer	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_P11Protect() is called, it increments the counter up to 0xFF.

⌋

6.12.4.2 Profile 11 Check Status Type

[PRS_E2E_00662]

Upstream requirements: RS_E2E_08528

⌈The E2E_P11Check functions 'State' shall have the members defined in [PRS_E2E_00891].⌋

[PRS_E2E_00891] E2E Profile 11 Check Status Type

Upstream requirements: RS_E2E_08528

⌈

Member Name	Type	Description
Counter	Unsigned Integer	Counter of the data in previous cycle.
Status	Enumeration	Result of the verification of the Data in this cycle, determined by the Check function.

⌋

6.12.4.3 Profile 11 Check Status Enumeration

[PRS_E2E_00594]

Upstream requirements: RS_E2E_08528

⌈The E2E_P11Check functions 'State->Status' shall consist of the following enumeration values (see [PRS_E2E_00892]).⌋

[PRS_E2E_00892] E2E Profile 11 Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_P11STATUS_OK	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
E2E_P11STATUS_NONEWDATA	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_P11STATUS_REPEATED.
E2E_P11STATUS_ERROR	Error	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
E2E_P11STATUS_REPEATED	Error	The checks of the Data in this cycle were successful, with the exception of the repetition.
E2E_P11STATUS_OKSOMELOST	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	Error	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta

⌋

6.12.4.4 Profile 11 Configuration Type

[PRS_E2E_00663]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P11Protect, E2E_P11Forward and E2E_P11Check functions 'Config' shall have the following members defined in [PRS_E2E_00893].⌋

[PRS_E2E_00893] E2E Profile 11 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
CounterOffset	Unsigned Integer	Bit offset of Counter in MSB first order. In variants 1A and 1B, CounterOffset is 8. The offset shall be a multiple of 4.
CRCOffset	Unsigned Integer	Bit offset of CRC (i.e. since *Data) in MSB first order. The offset shall be a multiple of 8. In variants 11A and 11C, CRCOffset is 0.
DataID	Unsigned Integer	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	Unsigned Integer	Bit offset of the low nibble of the high byte of Data ID.
DataIDMode	Enumeration	Inclusion mode of ID in CRC computation (both bytes, alternating, or low byte only of ID included).
MaxDeltaCounter	Unsigned Integer	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.
DataLength	Unsigned Integer	Length of data, in bits. The value shall be a multiple of 8 and shall be <= 256.

⌋

6.12.5 E2E Profile 11 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P11ConfigType field	Value
CounterOffset	8
CRCOffset	0
DataID	0x123
DataIDNibbleOffset	12
DataIDMode	E2E_P11DATAID_BOTH
DataLength	64
MaxDeltaCounter	1
MaxNoNewOrRepeatedData	15
SyncCounterInit	0

Table 6.46: E2E Profile 11 protocol example configuration

E2E_P11ProtectStateType field	Value

Counter	0

Table 6.47: E2E Profile 11 example state initialization

Byte							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0xcc	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.48: E2E Profile 11 protect result DataIDMode = E2E_P11DATAID_BOTH, counter 0

Result data of E2E_P11Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0x91	0x01	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.49: E2E Profile 11 protect result DataIDMode = E2E_P11DATAID_BOTH, counter 1

6.12.5.1 DataIDMode set to E2E_P11DATAID_NIBBLE

Result data of E2E_P11Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0x2a	0x10	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.50: E2E Profile 11 protect result DataIDMode = E2E_P11DATAID_NIBBLE, counter 0

Result data of E2E_P11Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0x77	0x11	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.51: E2E Profile 11 protect result DataIDMode = E2E_P11DATAID_NIBBLE, counter 1

6.12.5.2 DataIDMode set to E2E_P11DATAID_NIBBLE, Offset set to 64

This is a typical use-case for using P11 with SOME/IP serializer, which puts an 8 byte header in front of the serialized user data. The CRC calculation includes the 8 byte header and then the payload data, excluding the CRC byte itself. “Offset 64” means

CRCOffset set to 64, CounterOffset set to 72, DataIDNibbleOffset set to 76. Result data of E2E_P11Protect() with data equals all zeros (0x00), counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	8	9	10	11	12	13	14	15
Data	0x7d	0x10	0x00	0x00	0x00	0x00	0x00	0x00
Field	*CRC*	*DataID-Nibble /Counter*	*Data*					

Table 6.52: E2E Profile 11 example protect result with short data and SOME/IP

6.13 Specification of E2E Profile 22

[PRS_E2E_00522]

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈Profile 22 shall provide the following control fields, transmitted at runtime together with the protected data: Counter, CRC, Data ID (see [PRS_E2E_00894]).⌋

[PRS_E2E_00894] E2E Profile 22 mechanisms

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549

⌈

Control field	Description
Counter	4 bits. (explicitly sent)
CRC	8 bits, polynomial in normal form 0x2F (Autosar notation), provided by CRC library. (explicitly sent)
Data ID List	16 8 bits values, linked to Counter value. Effectively 16 different values, one for each counter value. The Data ID List shall be unique system-wide.

⌋

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

E2E Mechanism	Detected communication faults
Counter	Repetition, loss, insertion, incorrect sequence, blocking
Transmission on a regular basis and timeout monitoring using E2E-Library [5]	Loss, delay, blocking

[5] Implementation by sender and receiver

Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, asymmetric information [6]

Table 6.53: Detectable communication faults using Profile 22

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P22LENGTH_POS	0
P22LENGTH_LEN	0
P22COUNTER_POS	1
P22COUNTER_LEN	1 (only 4 bits are used, the maximum counter value is 15)
P22DATAID_POS	0
P22DATAID_LEN	0
P22CRC_POS	0
P22CRC_LEN	1
P22CALCULATE_CRC	Crc_CalculateCRC8H2F()

Table 6.54: Profile 22-specific data

For behavior and flowcharts of E2E Profile 22 see Chapter 6.3.

6.13.1 Header layout

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

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

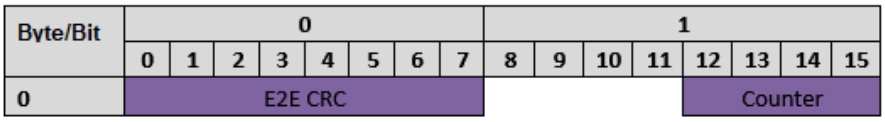

Figure 6.113: E2E Profile22 header with offset 0

The Figure 6.113 above shows Profile 22 with offset configured with 0. Offset is always given in bit and a multiple of 8 (full bytes).

[6] for a set of data protected by same CRC

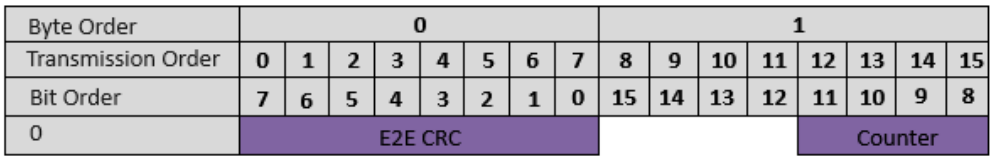

Figure 6.114: E2E Profile22 header with offset 0 and bit order MSB

For comparability to the figures of profile 2 in Figure 6.114 the bit order is added. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

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

6.13.1.1 Counter

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

[PRS_E2E_00523]

Upstream requirements: RS_E2E_08539

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

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

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

6.13.1.2 Data ID

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

[PRS_E2E_00524]

Upstream requirements: RS_E2E_08539

⌈In the E2E Profile 22, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation.⌋

[PRS_E2E_00525]

Upstream requirements: RS_E2E_08539

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

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

In case of usage of E2E Supervision for protecting messages (i.e. invocation from COM), the receiver COM expects at a reception only a specific message, which is checked by E2E Supervision using Data ID.

6.13.1.3 Length

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

6.13.1.4 CRC

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

[PRS_E2E_00526]

Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E Profile 22 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Supervision for calculating the CRC (Polynomial 0x2F, see also SWS_E2E_00117)⌋

[PRS_E2E_00527]

Upstream requirements: RS_E2E_08539, RS_E2E_08531

⌈In E2E Profile 22, the CRC shall be calculated over the entire E2E header (excluding the CRC byte), including the user data extended at the end with the corresponding Data ID from the Data ID List.⌋

6.13.2 Creation of E2E-Header

6.13.2.1 E2E_P22Protect

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

[PRS_E2E_00528]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P22Protect() shall have the overall behavior as shown in Figure 6.115.⌋

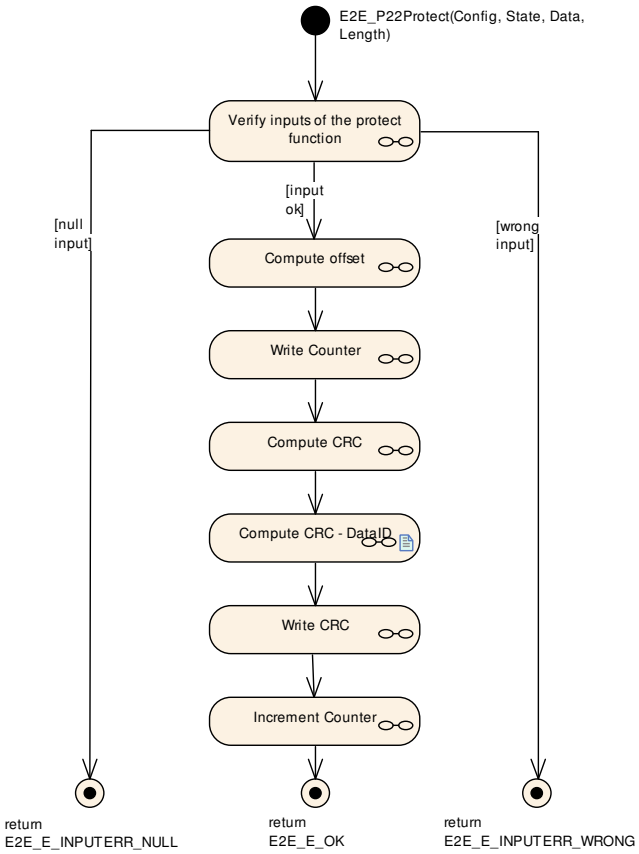

Figure 6.115: E2E Profile 22 Protect

[PRS_E2E_00529]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_P22Protect() shall behave as shown in Figure 6.3.⌋

[PRS_E2E_01253] Compute Offset of Protect Function

Upstream requirements: RS_E2E_08539

⌈The step ”Compute Offset of the protect function” in E2E_P22Protect(), E2E_P22Forward() and E2E_P22Check() shall behave as shown in Figure 6.4.⌋

[PRS_E2E_00530]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P22Protect() and E2E_P22Forward() shall behave as shown in Figure 6.116.⌋

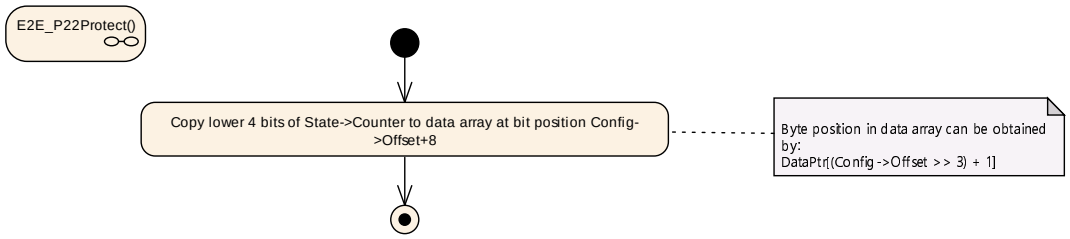

Figure 6.116: E2E Profile 22 Protect step ”Write Counter”

[PRS_E2E_00531]

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P22Protect(), E2E_P22Forward() and in E2E_P22Check shall behave as shown in Figure 6.8.⌋

[PRS_E2E_01254] Compute CRC - Data ID (Protect and Check Function)

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC - Data ID” in E2E_P22Protect() and E2E_P22Check() shall behave as shown in Figure 6.117.⌋

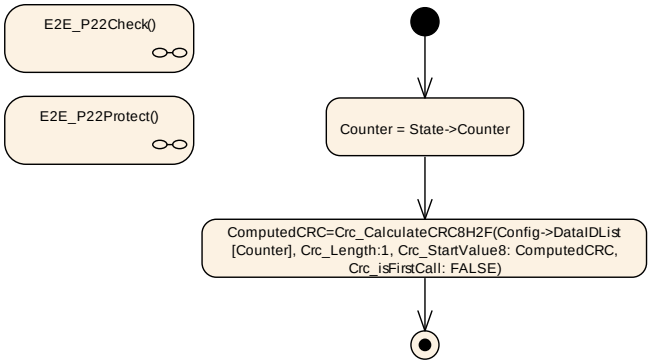

Figure 6.117: E2E Profile 22 Protect and Check step ”Compute CRC - Data ID”

[PRS_E2E_00532]

Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC” in E2E_P22Protect() and E2E_P22Forward() shall behave as shown in Figure 6.9.⌋

[PRS_E2E_00533]

Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_P22Protect() and E2E_P22Forward() shall behave as shown in Figure 6.10.⌋

6.13.2.2 E2E_P22Forward

The E2E_P22Forward() function of E2E Profile 22 is called by a SW-C in order to protect its application data and forward an received E2E-Status for use cases like translation of signal based to service oriented communication. If the received E2E status equals E2E_P_OK the behavior of the function shall be the same like E2E_P22Protect(). The function E2E_P22Forward() performs the steps as specified by the following diagrams in this section.

[PRS_E2E_00635]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P22Forward() shall have the overall behavior as shown in Figure 6.118.⌋

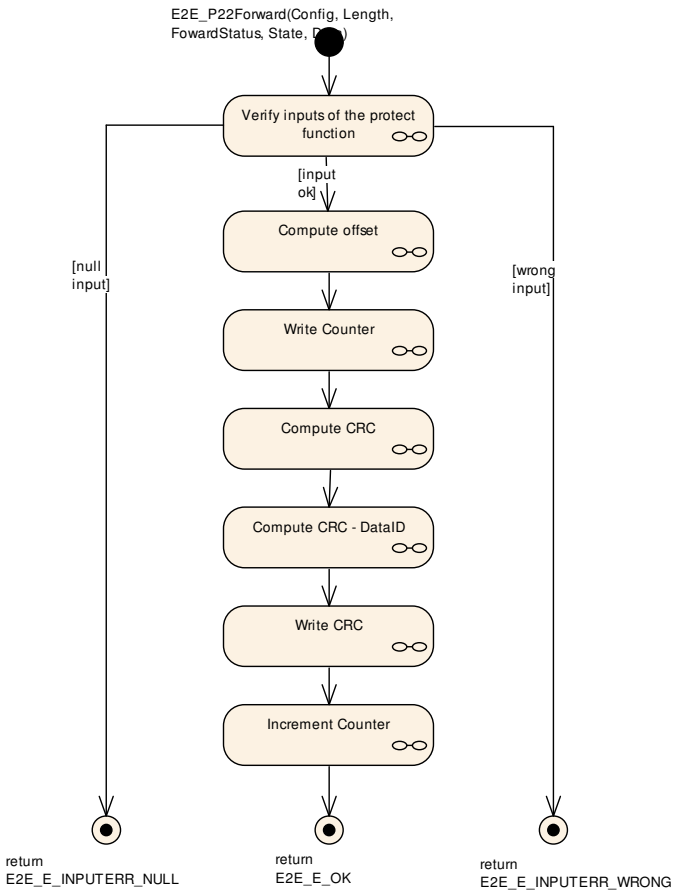

Figure 6.118: E2E Profile 22 Forward

Following steps are described in Section in Section 6.13.2.1

- ”Compute Offset” see [PRS_E2E_01253]
- ”Increment Counter” see [PRS_E2E_00533]
- ”Compute CRC” see [PRS_E2E_00531]
- ”Write CRC” see [PRS_E2E_00532]

[PRS_E2E_00636]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the forward function” in E2E_P22Forward() shall behave as shown in Figure 6.12.⌋

[PRS_E2E_00637]

Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P22Forward() shall behave as shown in Figure 6.119.⌋

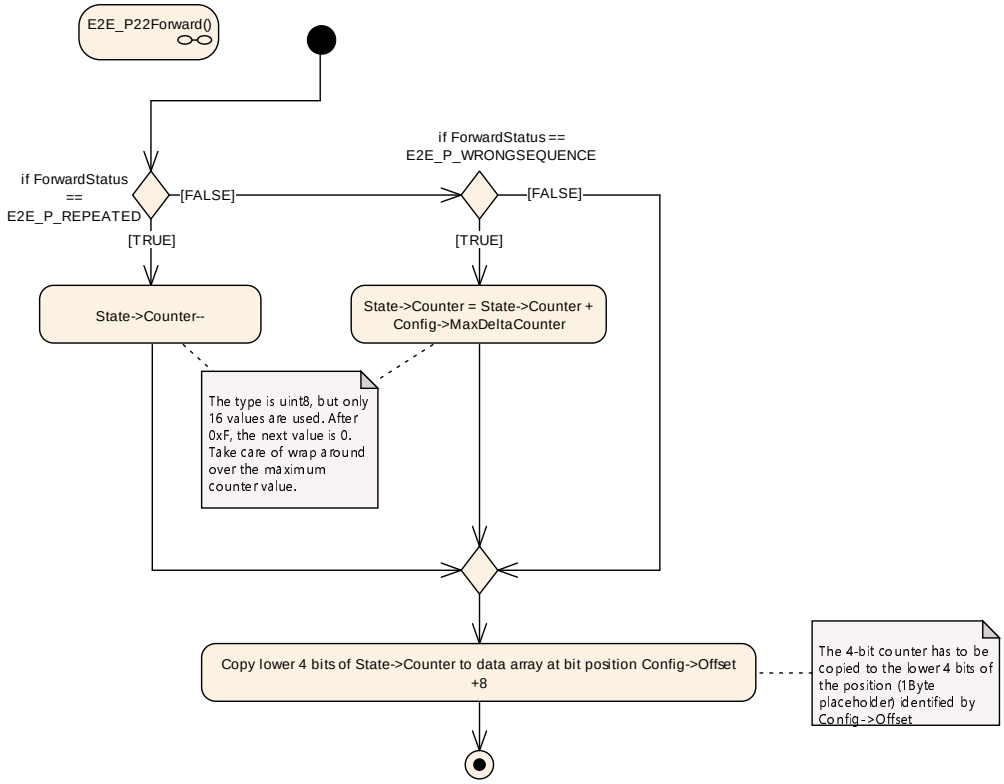

Figure 6.119: E2E Profile 22 Forward step ”Write Counter”

[PRS_E2E_01255] Compute CRC - Data ID (Forward Function)

Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC - Data ID” in E2E_P22Forward() shall behave as shown in Figure 6.120.⌋

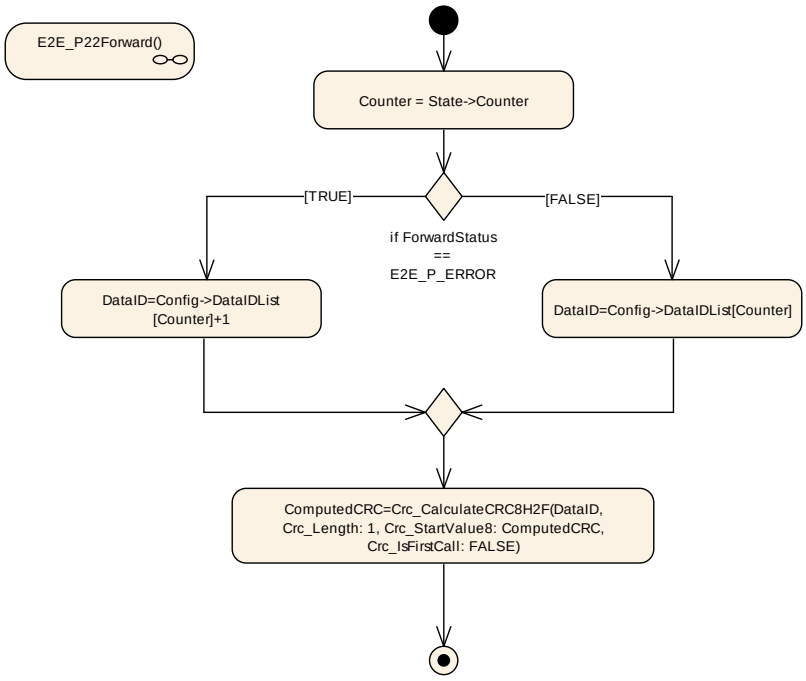

Figure 6.120: E2E Profile 22 Forward step ”Compute CRC - Data ID”

6.13.3 Evaluation of E2E-Header

6.13.3.1 E2E_P22Check

The function E2E_P22Check performs the actions as specified by the following diagrams in this section.

[PRS_E2E_00534]

Upstream requirements: RS_E2E_08539

⌈The function E2E_P22Check() shall have the overall behavior as shown in Figure 6.121.⌋

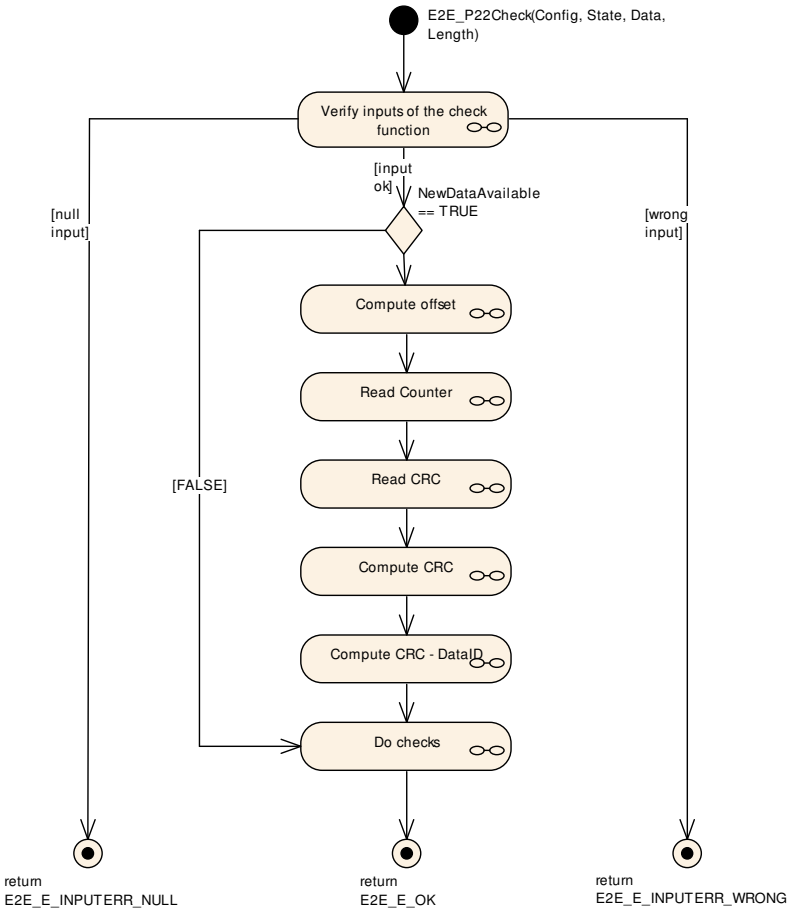

Figure 6.121: E2E Profile 22 Check

Following steps are described in Section in Section 6.13.2.1

- ”Compute Offset” see [PRS_E2E_01253]
- ”Compute CRC” see [PRS_E2E_00531]
- ”Compute CRC - Data ID” see [PRS_E2E_01254]

[PRS_E2E_00535]

Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the check function” in E2E_P22Check() shall behave as shown in Figure 6.16.⌋

[PRS_E2E_00536]

Upstream requirements: RS_E2E_08539

⌈The step ”Read Counter” in E2E_P22Check() shall behave as shown in Figure 6.122.⌋

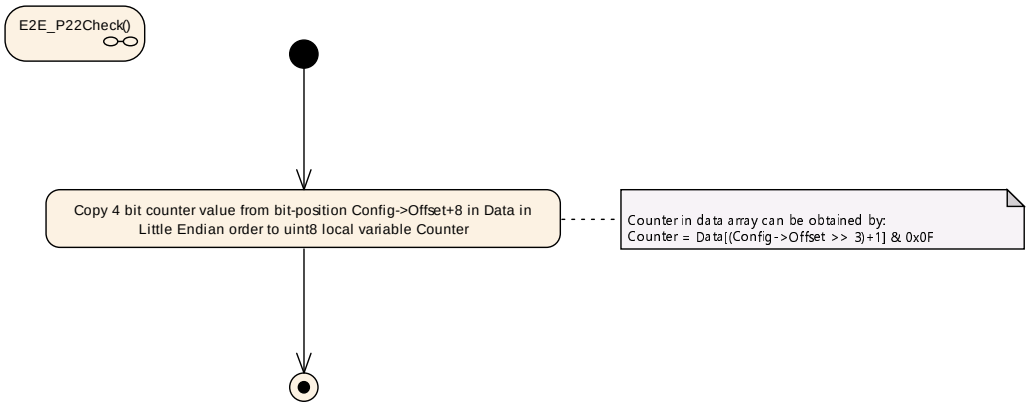

Figure 6.122: E2E Profile 22 Check step ”Read Counter”

[PRS_E2E_00537]

Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_P22Check() shall behave as shown in Figure 6.20.⌋

[PRS_E2E_00539]

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks’ in E2E_P22Check() shall behave as shown in Figure 6.21.⌋

6.13.4 Profile 22 Data Types

6.13.4.1 Profile 22 Configuration Type

[PRS_E2E_00666]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P22Protect, E2E_P22Forward and E2E_P22Check functions ’Config’ shall have the following members defined in [PRS_E2E_00898].⌋

[PRS_E2E_00898] E2E Profile 22 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
MinDataLength	Unsigned Integer	Length of Data, in bits. The value shall be a multiple of 8. It shall be : MinDataLength >= Offset+(2*8). The value must be the same as MaxDataLength.
MaxDataLength	Unsigned Integer	Length of Data, in bits. The value shall be a multiple of 8. In Profile 22 the value must be the same as MinDataLength.
DataIDList	Unsigned Integer Array	An array of appropriately chosen Data IDs for protection against masquerading.

MaxDeltaCounter	Unsigned Integer	Initial maximum allowed gap between two counter values of two consecutively received valid Data.
Offset	Unsigned Integer	Offset of the E2E header in the Data[] array in bits. It shall be: 0 <= Offset <= DataLength-(2*8).

⌋

6.13.5 E2E Profile 22 Protocol Examples

E2E_P22ConfigType field	Value
MinDataLength, MaxDataLength, DataLength	64
DataIDList	0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10
MaxDeltaCounter	1
MaxNoNewOrRepeatedData	15
SyncCounterInit	0
Offset	0

Table 6.55: E2E Profile 22 protocol example configuration

E2E_P22ProtectStateType field	Value
Counter	0

Table 6.56: E2E Profile 22 example state initialization

Result data of E2E_P22Protect() with data equals all zeros (0x00), counter starting with 0:

Counter	Byte							
	0	1	2	3	4	5	6	7
0	0x0e	0x00	0x00	0x00	0x00	0x00	0x00	0x00
1	0x1b	0x01	0x00	0x00	0x00	0x00	0x00	0x00
2	0x98	0x02	0x00	0x00	0x00	0x00	0x00	0x00
3	0x31	0x03	0x00	0x00	0x00	0x00	0x00	0x00
4	0x0d	0x04	0x00	0x00	0x00	0x00	0x00	0x00
5	0x18	0x05	0x00	0x00	0x00	0x00	0x00	0x00
6	0x9b	0x06	0x00	0x00	0x00	0x00	0x00	0x00
7	0x65	0x07	0x00	0x00	0x00	0x00	0x00	0x00
8	0x08	0x08	0x00	0x00	0x00	0x00	0x00	0x00
9	0x1d	0x09	0x00	0x00	0x00	0x00	0x00	0x00
10	0x9e	0x0a	0x00	0x00	0x00	0x00	0x00	0x00
11	0x37	0x0b	0x00	0x00	0x00	0x00	0x00	0x00
12	0x0b	0x0c	0x00	0x00	0x00	0x00	0x00	0x00

▽

△

Counter	Byte							
	0	1	2	3	4	5	6	7
13	0x1e	0x0d	0x00	0x00	0x00	0x00	0x00	0x00
14	0x9d	0x0e	0x00	0x00	0x00	0x00	0x00	0x00
15	0xcd	0x0f	0x00	0x00	0x00	0x00	0x00	0x00

Table 6.57: E2E Profile 22 example protect result

6.13.5.1 Offset set to 64

This is a typical use-case for using P22 with SOME/IP serializer, which puts an 8 byte header in front of the serialized user data. Result data of E2E_P22Protect() with data equals all zeros (0x00), counter = 1:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	8	9	10	11	12	13	14	15
Data	0x14	0x01	0x00	0x00	0x00	0x00	0x00	0x00
Field	*CRC*	*DataID /Counter*	*Data*					

Table 6.58: E2E Profile 22 example protect result with short data and SOME/IP

6.14 Specification of E2E Profile 44

[PRS_E2E_00707]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈Profile 44 shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID (see [PRS_E2E_00906]).⌋

[PRS_E2E_00906] E2E Profile 44 mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈

Control field	Description
Length	16 bits, to support dynamic-size data.
Counter	16-bits.
CRC	32 bits, polynomial in normal form 0xF4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCP/IP.
Data ID	32-bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P44LENGTH_POS	0
P44LENGTH_LEN	2
P44COUNTER_POS	2
P44COUNTER_LEN	2
P44DATAID_POS	4
P44DATAID_LEN	4
P44CRC_POS	8
P44CRC_LEN	4
P44CALCULATE_CRC	Crc_CalculateCRC32P4()

Table 6.59: Profile 44-specific data

For behavior and flowcharts of E2E Profile 44 see Chapter 6.3.

6.14.1 Header Layout

In the E2E Profile 44, the user data layout (of the data to be protected) is not constrained by E2E Profile 44 - there is only a requirement that the length of data to be protected is multiple of 1 byte.

The header of the E2E Profile 44 has one fixed layout, as follows:

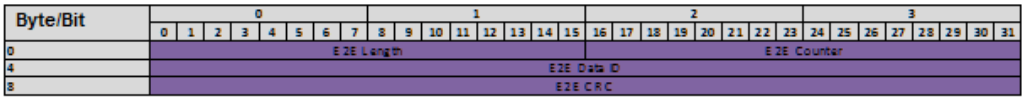

Figure 6.123: E2E Profile 44 Header

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

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

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

6.14.2 Profile 44 Configuration Type

[PRS_E2E_00735]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P44Protect, E2E_P44Forward and E2E_P44Check functions 'Config' shall have the following members defined in [PRS_E2E_00907]. The current DataLength shall be a multiple of 8 as well as the MaxDataLength and MinDataLength.⌋

[PRS_E2E_00907] E2E Profile 44 Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Member Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be a multiple of 8. The value shall be >= 12*8 and <= MaxDataLength. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 65535*8 and >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.14.3 E2E Profile 44 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P44ConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000 0000
MinDataLength	96
MaxDataLength	524280

MaxDeltaCounter	1

Table 6.60: E2E Profile 44 protocol example configuration

E2E_P44ProtectStateType field	Value
Counter	0

Table 6.61: E2E Profile 44 example state initialization

Result data of E2E_P44Protect() with short data length (length 16 bytes, means 4 actual data bytes), offset = 0, counter = 0:

Byte	1	2	3	4	5	6	7	8
Data	0x00	0x10	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	9	10	11	12	13	14	15	16
Data	0x86	0x2b	0x05	0x56	0x00	0x00	0x00	0x00
Field	*CRC*				*Data*			

Table 6.62: E2E Profile 44 example short

Result data of E2E_P44Protect() with minimum data length (4 data bytes), offset = 64 (as with SOME/IP header use case), datalength = 24, counter = 0:

Byte	1	2	3	4	5	6	7	8
Data	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Field	*Data (upper header)*							
Byte	9	10	11	12	13	14	15	16
Data	0x00	0x18	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	17	18	19	20	21	22	23	24
Data	0x69	0xd7	0x50	0x2e	0x00	0x00	0x00	0x00
Field	*CRC*				*Data*			

Table 6.63: E2E Profile 44 example short with SOME/IP use case

6.15 Specification of E2E Profile 76

[PRS_E2E_01318] Profile 76 Mechanisms

Status: DRAFT

Upstream requirements: RS_E2E_08527, RS_E2E_08529, RS_E2E_08530, RS_E2E_08533, RS_E2E_08543, RS_E2E_08544, RS_E2E_08545, RS_E2E_08546, RS_E2E_08547, RS_E2E_08548, RS_E2E_08549, RS_E2E_08550

⌈Profile 76 shall provide the following control fields, transmitted at runtime together with the protected data: Counter, CRC (see Table 6.64).⌋

Control field	Description
Counter	5 bits.
CRC	32 bits, polynomial in normal form 0x6938392D, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCP/IP.
Data ID	This profile does not use Data ID in the profile header and not as input for CRC calculation.

Table 6.64: E2E Profile 76 Control Fields

P76 in the current version does not use an E2E Data ID. The reason is that SAE J1939-76 currently does not require the CRC to be calculated over the PGN and the source address, which together have the role of a Data ID. But the underlying J1939Fscp module ensures a correct sequence of SHM and SDM, so that most masquerading and addressing errors can still be detected.

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P76COUNTER_POS	0
P76COUNTER_LEN	1
P76CRC_POS	1
P76CRC_LEN	4
P76CALCULATE_CRC	Crc_CalculateCRC32_J1939()

Table 6.65: Profile 76 specific data

6.15.1 Header Layout

In the E2E Profile 76, the data to be protected is between 1 and 8 bytes payload data and the length of data to be protected is multiple of 1 byte.

The header of the E2E Profile 76 has one fixed layout, as follows:

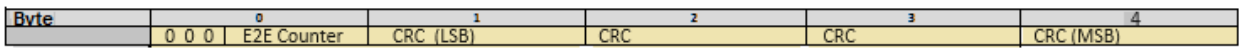

Figure 6.124: E2E Profile 76 Header

1. Little Endian (least significant byte first) - imposed by profile
2. E2E counter is stored in lower 5 bits. Higher 3 bits of byte are set to 0.

The header is placed upfront of the message.

6.15.1.1 Counter

In E2E Profile 76, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E Supervision.

[PRS_E2E_01400] Profile 76 Initialization of counter

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈In E2E Profile 76, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0x1F), then it shall restart with 0 for the next send request.⌋

6.15.1.2 Length

In Profile 76 there is no explicit transmission of the length.

6.15.1.3 CRC

E2E Profile 76 uses a 32-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[PRS_E2E_01401] Profile 76 CRC Calculation Algorithm

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08539

⌈E2E Profile 76 shall use the Crc_CalculateCRC SAE J1939-76 CRC function for calculating the CRC (Polynomial: 0X6938392D; Autosar notation).⌋

[PRS_E2E_01402] Profile 76 CRC Calculation over E2E Header

Status: DRAFT
Upstream requirements: RS_E2E_08539, RS_E2E_08531

⌈In E2E Profile 76, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended at the end.⌋

6.15.2 Creation of the E2E-Header

6.15.2.1 E2E_P76Protect

The function E2E_P76Protect() performs the steps as specified by the following diagrams.

[PRS_E2E_01420] Profile 76 Protect Function

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The function E2E_P76Protect() shall have the overall behavior as shown in Figure 6.125

⌋

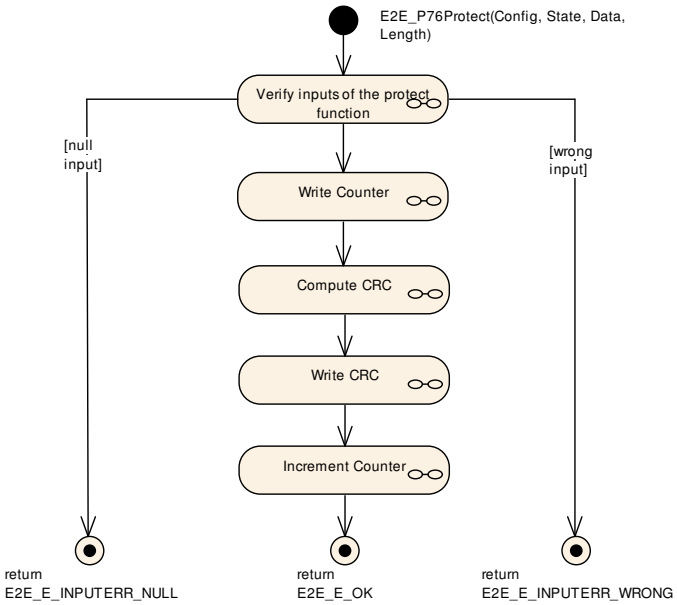

Figure 6.125: E2E Profile 76 Protect

[PRS_E2E_01422] Profile 76 Verification of Inputs of Protect Function

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the protect function” in E2E_P76Protect() shall behave as shown in Figure 6.3

⌋

[PRS_E2E_01424] Profile 76 Write Counter

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Write Counter” in E2E_P76Protect() shall behave as shown in Figure 6.126⌋

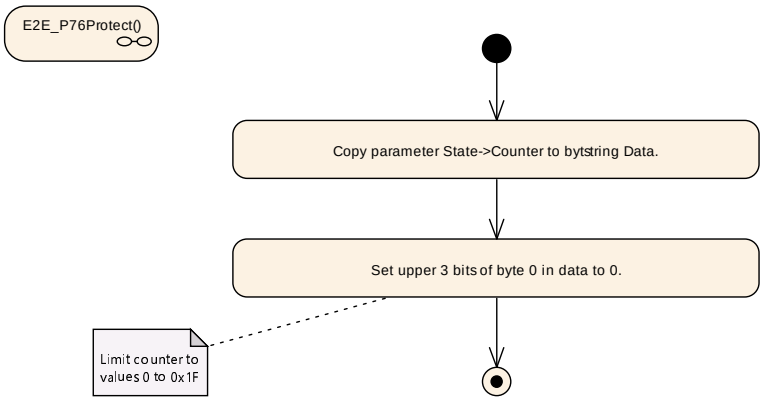

Figure 6.126: E2E Profile 76 Protect step ”Write Counter”

[PRS_E2E_01427] Profile 76 Compute CRC

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P76Protect() and in E2E_P76Check shall behave as shown in Figure 6.127⌋

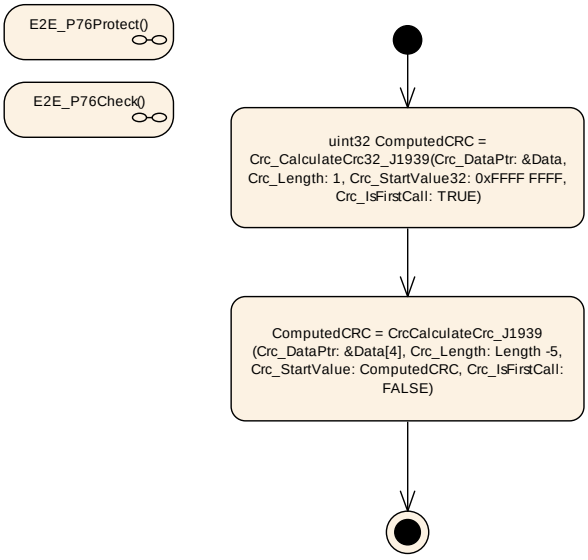

Figure 6.127: E2E Profile 76 Protect and Check step ”Compute CRC”

[PRS_E2E_01429] Profile 76 Write CRC

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Write CRC” in E2E_P76Protect() shall behave as shown in Figure 6.9⌋

[PRS_E2E_01430] Profile 76 Increment Counter

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Increment Counter” in E2E_P76Protect() shall behave as shown in Figure 6.128⌋

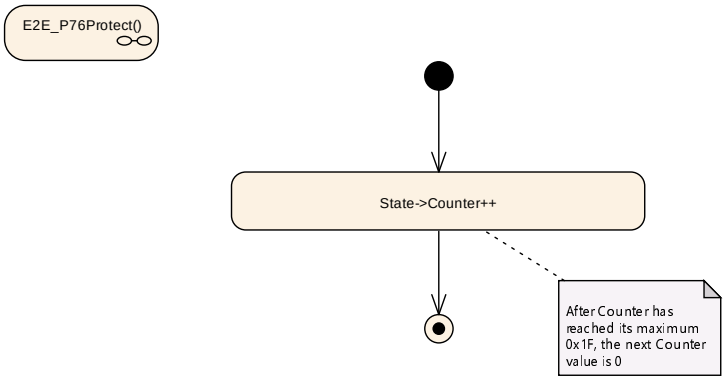

Figure 6.128: E2E Profile 76 Protect step ”Increment Counter”

6.15.3 Evaluation of the E2E-Header

6.15.3.1 E2E_P76Check

The function E2E_P76Check() performs the actions as specified by the following diagrams.

[PRS_E2E_01403] Profile 76 Check function

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The function E2E_P76Check() shall have the overall behavior as shown in Figure 6.129⌋

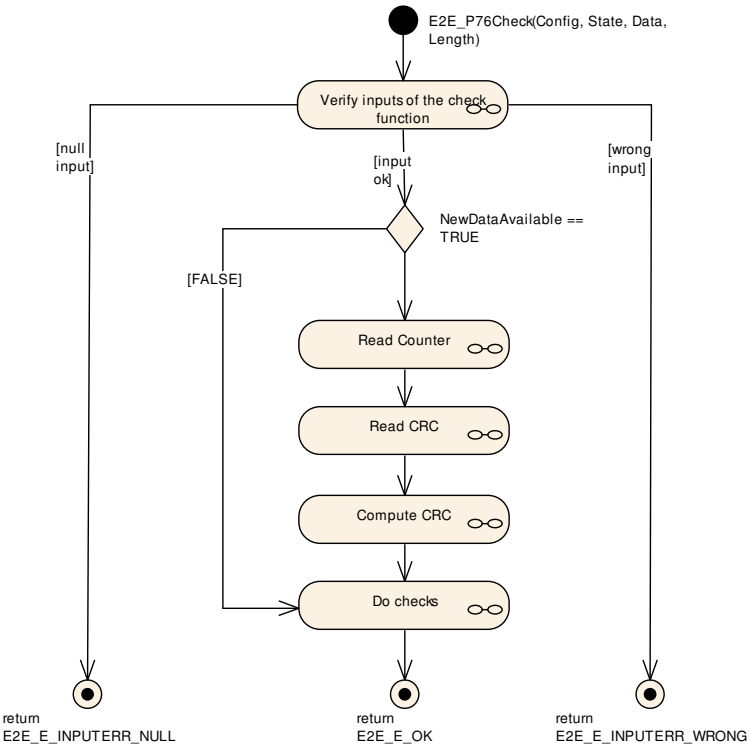

Figure 6.129: E2E Profile 76 Check

[PRS_E2E_01405] Profile 76 Verification of inputs of Check function

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Verify inputs of the check function” in E2E_P76Check() shall behave as shown in Figure 6.16⌋

[PRS_E2E_01406] Profile 76 Read Counter

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Read Counter” in E2E_P76Check() shall behave as shown in Figure 6.18⌋

[PRS_E2E_01407] Profile 76 Read CRC

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Read CRC” in E2E_P76Check() shall behave as shown in Figure 6.20⌋

[PRS_E2E_01433] Profile 76 Compute CRC

Status: DRAFT
Upstream requirements: RS_E2E_08539

⌈The step ”Compute CRC” in E2E_P76Check() shall behave as shown in Figure 6.127.⌋

[PRS_E2E_01408] Profile 76 Perform checks

Status: DRAFT

Upstream requirements: RS_E2E_08539

⌈The step ”Do Checks’ in E2E_P76Check() shall behave as shown in Figure 6.130.⌋

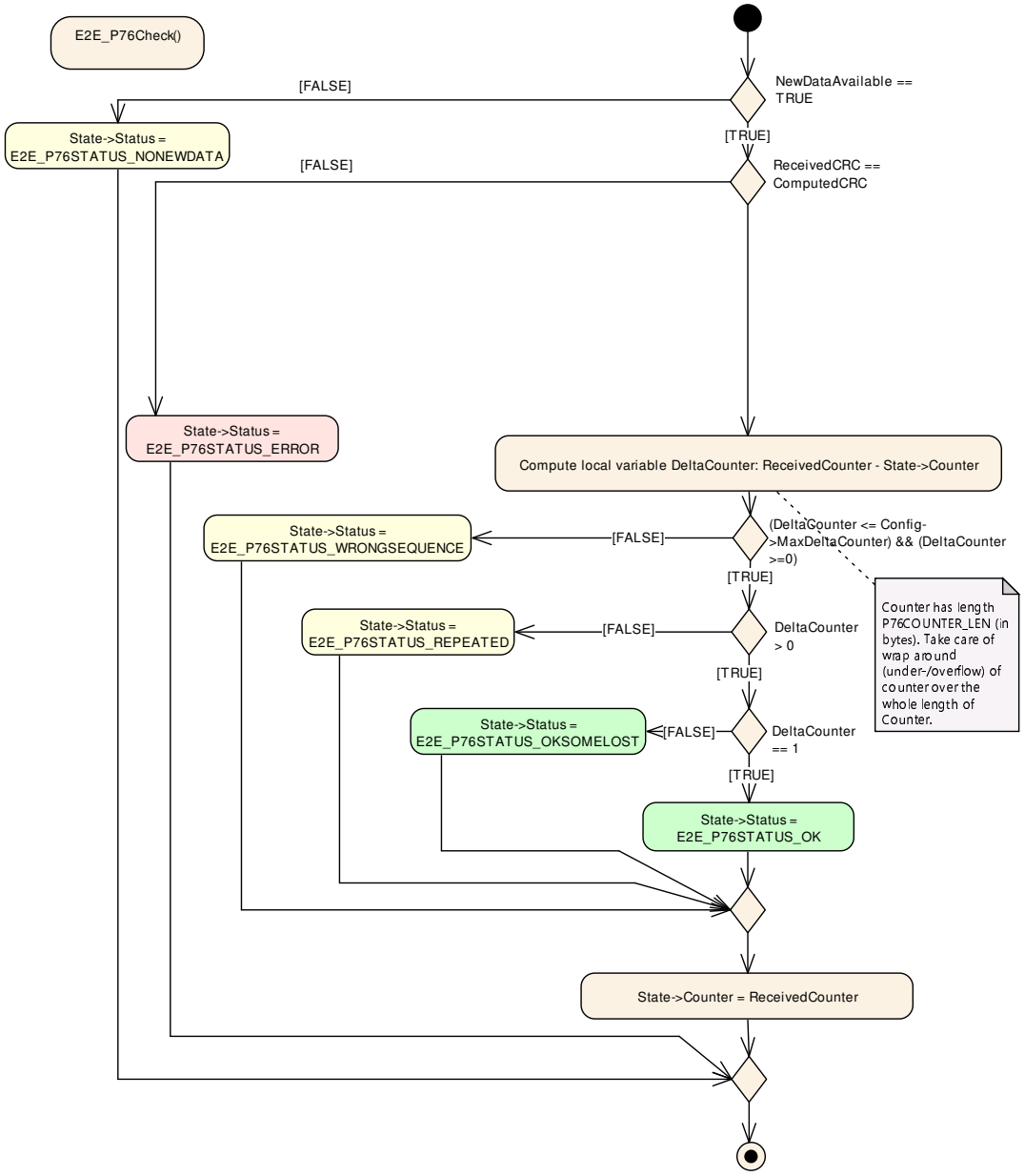

Figure 6.130: E2E Profile 76 Do Checks

6.15.4 Profile Data Types

6.15.4.1 Profile 76 Protect State Type

[PRS_E2E_01409] Profile 76 Protect Function

Status: DRAFT
Upstream requirements: RS_E2E_08528

⌈The E2E_P76Protect function 'state' shall have the members defined in [PRS_E2E_01410].⌋

[PRS_E2E_01410] E2E Profile 76 Protect State Type

Status: DRAFT
Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
Counter	Unsigned Integer	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_P76Protect() is called, it increments the counter up to 0x1F.

⌋

6.15.4.2 Profile 76 Check State Type

[PRS_E2E_01411] Profile 76 Check Function

Status: DRAFT
Upstream requirements: RS_E2E_08528

⌈The E2E_P76Check functions 'State' shall have the members defined in [PRS_E2E_01412].⌋

[PRS_E2E_01412] E2E Profile 76 Check State Type

Status: DRAFT
Upstream requirements: RS_E2E_08528

⌈

Member Name	State Type	Description
Counter	Unsigned Integer	Counter of the data in previous cycle.
Status	Enumeration	Result of the verification of the Data in this cycle, determined by the Check function.

⌋

6.15.4.3 Profile 76 Check Status Enumeration

[PRS_E2E_01413] Profile 76 Status

Status: DRAFT
Upstream requirements: RS_E2E_08528

⌈The E2E_P76Check functions 'State->Status' enumeration type shall consist the following enumeration values (see [PRS_E2E_01414]).⌋

[PRS_E2E_01414] E2E Profile 76 Check Status Enumeration

Status: DRAFT
Upstream requirements: RS_E2E_08528

⌈

Name	State Type	Description
E2E_P76STATUS_OK	OK	The checks of the Data in this cycle were successful (including counter check, which was incremented by 1).
E2E_P76STATUS_NONEWDATA	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_P76STATUS_REPEATED.
E2E_P76STATUS_ERROR	Error	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong options, wrong Data ID).
E2E_P76STATUS_REPEATED	Error	The checks of the Data in this cycle were successful, with the exception of the repetition.
E2E_P76STATUS_OKSOMELOST	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	Error	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta

⌋

6.15.4.4 Profile 76 Configuration Type

[PRS_E2E_01415] Profile 76 Configuration

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08537, RS_E2E_08539

⌈The E2E_P76Protect and E2E_P76Check functions 'Config' shall have the following members defined in [PRS_E2E_01416].⌋

[PRS_E2E_01416] E2E Profile 76 Configuration Type

Status: DRAFT
Upstream requirements: RS_E2E_08528, RS_E2E_08537, RS_E2E_08539

⌈

Member Name	Type	Description
MinDataLength	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be <= MaxDataLength and shall be >= 6*8. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 13*8 (13 byte) and it shall be >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.15.5 E2E Profile 76 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P76ConfigType field	Value
MinDataLength	48
MaxDataLength	104

Table 6.66: E2E Profile 76 protocol example configuration

E2E_P76ProtectStateType field	Value
Counter	0

Table 6.67: E2E Profile 76 example state initialization

Result data of E2E_P76Protect() with full data length (length 13 bytes, means 8 actual data bytes, f.e. 0x01 0x02 0x03 .. 0x07), counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0xC5	0x50	0x53	0x7D	0x00	0x01	0x02
Field	*Counter*	*CRC*				*Payload*		
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0x03	0x04	0x05	0x06	0x07			
Field	*Payload*					*unused*		

Table 6.68: E2E Profile 76 example short with 8 bytes payload

Result data of E2E_P76Protect() with full data length (length 13 bytes, means 8 actual data bytes, f.e. 0x12 0x34 0x56 ... 0xDE 0xF0), counter = 0:

Byte	0	1	2	3	4	5	6	7
Data	0x00	0xD7	0x71	0x3A	0x27	0x12	0x34	0x56
Field	*Counter*	*CRC*				*Payload*		
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0x87	0x9A	0xBC	0xDE	0xF0			
Field	*Payload*					*unused*		

Table 6.69: E2E Profile 76 example short with 8 bytes payload

6.16 Specification of E2E Profile 4m

[PRS_E2E_00740]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈Profile 4m shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID, Source ID, Message Type, Message Result (see [PRS_E2E_00899]).⌋

[PRS_E2E_00899] E2E Profile 4m mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈

Control field	Description
Length	16 bits, to support dynamic-size data.
Counter	16 bits.
CRC	32 bits, polynomial in normal form 0xF4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCP/IP.
Data ID	32 bits, unique system-wide.
Message Type	2-bits, request (0) vs. response (1).
Message Result	2-bits, OK (0) vs. error (1) - fixed to OK (0) for Message Type 'request'.
Source ID	28-bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P04MLENGTH_POS	0
P04MLENGTH_LEN	2
P04MCOUNTER_POS	2
P04MCOUNTER_LEN	2
P04MDATAID_POS	4
P04MDATAID_LEN	4
P04MCRC_POS	8
P04MCRC_LEN	4
P04MMESSAGE_POS	12
P04MCALCULATE_CRC	Crc_CalculateCRC32P4()

Table 6.70: Profile 4m-specific data

For behavior and flowcharts of E2E method Profile 4m see Chapter 6.4.

6.16.1 Header Layout

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

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

Byte/Bit	0								1								2								3							
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	E2E Length																E2E Counter															
4	E2E Data ID																															
8	E2E CRC																															
12	T		R		E2E Source ID																											

Table 6.71: E2E Profile 4m Header

Hereby ‘T’ denotes the E2E Message Type and ‘R’ denotes the E2E Message Result. The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte first) - imposed by profile
2. LSB First (least significant bit within byte first) - imposed by TCP/IP network

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

6.16.2 Profile 4m Configuration Type

[PRS_E2E_00852]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P04mProtect, E2E_P04mForward, E2E_P04mSourceCheck and E2E_P04mSinkCheck functions "Config" shall have the following members defined in [PRS_E2E_00900].⌋

[PRS_E2E_00900] E2E Profile 4m Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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 (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
MinDataLength	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be <= MaxDataLength and shall be >= 16*8. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 4096*8 (4KB) and it shall be >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.16.3 E2E Profile 4m Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P04mConfigType field	Value

DataID	0x0a0b0c0d
Offset	0x0000
MinDataLength	128
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.72: E2E Profile 4m protocol example configuration

E2E_P04mProtectStateType field	Value
Counter	0

Table 6.73: E2E Profile 4m example state initialization

6.16.4 Request Example

The result data of an E2E_P04mProtect() call with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_REQUEST, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.74.

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data	0x00	0x14	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0xae	0x67	0x4c	0xa0	0x00	0x12	0x34	0x56
Field	*CRC*				*Message Type*, *Message Result*, *Source ID*			
Byte	**16**	**17**	**18**	**19**	**n/a**			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.74: E2E Profile 4m example short - request (Message Type = STD_MESSAGETYPE_REQUEST; Message Result = STD_MESSAGERESULT_OK)

6.16.5 Response Example

The result data of an E2E_P04mProtect() call with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.75.

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data	0x00	0x14	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			

▽

△

Byte	8	9	10	11	12	13	14	15
Data	0x85	0x25	0x76	0x19	0x40	0x12	0x34	0x56
Field	*CRC*				*Message Type, Message Result, Source ID*			
Byte	**16**	**17**	**18**	**19**	**n/a**			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.75: E2E Profile 4m example short - normal response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_OK)

6.16.6 Error Response Example

The result data of an E2E_P04mProtect() call with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_ERROR is depicted in Table 6.76.

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x14	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0x23	0x45	0x57	0x0f	0x50	0x12	0x34	0x56
Field	*CRC*				*Message Type, Message Result, Source ID*			
Byte	**16**	**17**	**18**	**19**	**n/a**			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.76: E2E Profile 4m example short - ERROR reponse (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_ERROR)

6.17 Specification of E2E Profile 7m

[PRS_E2E_00783]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈Profile 7m shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID, Source ID, Message Type, Message Result (see [PRS_E2E_00903]).⌋

[PRS_E2E_00903] E2E Profile 7m mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈

Control field	Description
Length	32 bits, to support dynamic-size data.
Counter	32 bits.
CRC	64 bits, polynomial in normal form 0x42F0E1EBA9EA3693, provided by CRC library. Note: This CRC polynomial is also known as “CRC-64 (ECMA)”.
Data ID	32 bits, unique system-wide.
Message Type	2-bits, request (0) vs. response (1).
Message Result	2-bits, OK (0) vs. error (1) - fixed to OK (0) for Message Type ‘request’.
Source ID	28-bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P07MLENGTH_POS	8
P07MLENGTH_LEN	4
P07MCOUNTER_POS	12
P07MCOUNTER_LEN	4
P07MDATAID_POS	16
P07MDATAID_LEN	4
P07MCRC_POS	0
P07MCRC_LEN	8
P07MMESSAGE_POS	20
P07MCALCULATE_CRC	Crc_CalculateCRC64()

Table 6.77: Profile 8m-specific data

For behavior and flowcharts of E2E method Profile 7m see Chapter 6.4.

6.17.1 Header Layout

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

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

Byte/Bit	0								1								2								3							
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	E2E CRC																															
4	E2E CRC																															
8	E2E Length																															
12	E2E Counter																															
16	E2E Data ID																															
20	T		R		E2E Source ID																											

Table 6.78: E2E Profile 7m Header

Hereby ‘T’ denotes the E2E Message Type and ‘R’ denotes the E2E Message Result. The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte first) - imposed by profile
2. LSB First (least significant bit within byte first) - imposed by TCP/IP network

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

6.17.2 Profile 7m Configuration Type

[PRS_E2E_00851]

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈The E2E_P07mProtect, E2E_P07mForward, E2E_P07mSourceCheck and E2E_P07mSinkCheck functions “Config” shall have the following members defined in [PRS_E2E_00905].⌋

[PRS_E2E_00905] E2E Profile 7m Configuration Type

Upstream requirements: RS_E2E_08528, RS_E2E_08537

⌈

Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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 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	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be <= MaxDataLength and it shall be >= 24*8. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be >= MinDataLength and it should be <= 4194304*8 (4MB). The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.17.3 E2E Profile 7m Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P07mConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000
MinDataLength	192
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.79: E2E Profile 7m protocol example configuration

E2E_P07mProtectStateType field	Value
Counter	0

Table 6.80: E2E Profile 7m example state initialization

6.17.4 Request Example

The result data of an E2E_P07mProtect() call with short data length (length 28 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_REQUEST, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.81.

Byte	0	1	2	3	4	5	6	7
Data	0xae	0x96	0xa7	0xd0	0xa5	0x01	0x75	0x94
Field	*CRC*							

▽

△

Byte	8	9	10	11	12	13	14	15
Data	0x00	0x00	0x00	0x1c	0x00	0x00	0x00	0x00
Field	*Length*				*Counter*			
Byte	16	17	18	19	20	21	22	23
Data	0x0a	0x0b	0x0c	0x0d	0x00	0x12	0x34	0x56
Field	*DataID*				*Message Type, Message Result, Source ID*			
Byte	24	25	26	27	n/a			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.81: E2E Profile 7m example short - request (Message Type = STD_MESSAGETYPE_REQUEST; Message Result = STD_MESSAGERESULT_OK)

6.17.5 Response Example

The result data of an E2E_P07mProtect() call with short data length (length 28 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.82.

Byte	0	1	2	3	4	5	6	7
Data	0xa6	0x2d	0x64	0x86	0xe8	0x3f	0x2c	0xaf
Field	*CRC*							
Byte	8	9	10	11	12	13	14	15
Data	0x00	0x00	0x00	0x1c	0x00	0x00	0x00	0x00
Field	*Length*				*Counter*			
Byte	16	17	18	19	20	21	22	23
Data	0x0a	0x0b	0x0c	0x0d	0x40	0x12	0x34	0x56
Field	*DataID*				*Message Type, Message Result, Source ID*			
Byte	24	25	26	27	n/a			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.82: E2E Profile 7m example short - normal response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_OK)

6.17.6 Error Response Example

The result data of an E2E_P07mProtect() call with short data length (length 28 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_ERROR is depicted in Table 6.83.

Byte	0	1	2	3	4	5	6	7
Data	0x09	0xd9	0xe8	0x0c	0x47	0x34	0x32	0x02
Field	*CRC*							
Byte	8	9	10	11	12	13	14	15
Data	0x00	0x00	0x00	0x1c	0x00	0x00	0x00	0x00
Field	*Length*				*Counter*			
Byte	16	17	18	19	20	21	22	23
Data	0x0a	0x0b	0x0c	0x0d	0x50	0x12	0x34	0x56
Field	*DataID*				*Message Type, Message Result, Source ID*			
Byte	24	25	26	27	n/a			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.83: E2E Profile 7m example short - ERROR response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_ERROR)

6.18 Specification of E2E Profile 8m

[PRS_E2E_01107]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈Profile 08m shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID (see [PRS_E2E_00908]).⌋

[PRS_E2E_00908] E2E Profile 8m mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈

Control field	Description
Length	32 bits, to support dynamic-size data.
Counter	32 bits.
CRC	32 bits, polynomial in normal form 0x1F4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC polynomials used by FlexRay, CAN and LIN and TCPIP.
Data ID	32 bits, unique system-wide.
Message Type	2-bits, request (0) vs. response (1).
Message Result	2-bits, OK (0) vs. error (1) - fixed to OK (0) for Message Type 'request'.
Source ID	28-bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P08MLENGTH_POS	4
P08MLENGTH_LEN	4
P08MCOUNTER_POS	8
P08MCOUNTER_LEN	4
P08MDATAID_POS	12
P08MDATAID_LEN	4
P08MCRC_POS	0
P08MCRC_LEN	4
P08MMESSAGE_POS	16
P08MCALCULATE_CRC	Crc_CalculateCRC32P4()

Table 6.84: Profile 8m-specific data

For behavior and flowcharts of E2E method Profile 8m see Chapter 6.4.

6.18.1 Header Layout

In the E2E Profile 8m, the user data layout (of the data to be protected) is not constrained by E2E Profile 8m - there is only a requirement that the length of data to be protected is multiple of 1 byte.

The header of the E2E Profile 8m has one fixed layout, as follows:

Byte/Bit	0								1								2								3							
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	E2E CRC																															
4	E2E Length																															
8	E2E Counter																															
12	E2E Data ID																															
16	T		R		E2E Source ID																											

Table 6.85: E2E Profile 8m Header

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

1. Big Endian (most significant byte first) - imposed by profile
2. LSB First (least significant bit within byte first) - imposed by TCPIP bus

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

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

6.18.2 Profile 8m Configuration Type

[PRS_E2E_01154]

Upstream requirements: RS_E2E_08528

⌈The E2E_P08mProtect, E2E_P08mForward, and E2E_P08mCheck functions "Config" shall have the following members defined in [PRS_E2E_00909].⌋

[PRS_E2E_00909] E2E Profile 8m Configuration Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be >= 20*8 and <= MaxDataLength. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 536870912*8 and >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.18.3 E2E Profile 8m Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P08mConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000
MinDataLength	160
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.86: E2E Profile 8m protocol example configuration

E2E_P08mProtectStateType field	Value
Counter	0

Table 6.87: E2E Profile 8m example state initialization

6.18.4 Request Example

The result data of an E2E_P08mProtect() call with short data length (length 24 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_REQUEST, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.88.

Byte	0	1	2	3	4	5	6	7
Data	0x6d	0xf0	0x5e	0xba	0x00	0x00	0x00	0x18
Field	*CRC*				*Length*			
Byte	8	9	10	11	12	13	14	15
Data	0x00	0x00	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Counter*				*DataID*			
Byte	16	17	18	19	20	21	22	23
Data	0x00	0x12	0x34	0x56	0x00	0x00	0x00	0x00
Field	*Message Type, Message Result, Source ID*				*Data*			

Table 6.88: E2E Profile 8m example short - request (Message Type = STD_MESSAGETYPE_REQUEST; Message Result = STD_MESSAGERESULT_OK)

6.18.5 Response Example

The result data of an E2E_P08mProtect() call with short data length (length 24 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.89.

Byte	0	1	2	3	4	5	6	7
Data	0x46	0xb2	0x64	0x03	0x00	0x00	0x00	0x18
Field	*CRC*				*Length*			
Byte	8	9	10	11	12	13	14	15
Data	0x00	0x00	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Counter*				*DataID*			
Byte	16	17	18	19	20	21	22	23
Data	0x40	0x12	0x34	0x56	0x00	0x00	0x00	0x00
Field	*Message Type, Message Result, Source ID*				*Data*			

Table 6.89: E2E Profile 8m example short - normal response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_OK)

6.18.6 Error Response Example

The result data of an E2E_P08mProtect() call with short data length (length 24 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_ERROR is depicted in Table 6.90.

Byte	0	1	2	3	4	5	6	7
Data	0xe0	0xd2	0x45	0x15	0x00	0x00	0x00	0x18
Field	*CRC*				*Length*			
Byte	8	9	10	11	12	13	14	15
Data	0x00	0x00	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Counter*				*DataID*			
Byte	16	17	18	19	20	21	22	23
Data	0x50	0x12	0x34	0x56	0x00	0x00	0x00	0x00
Field	*Message Type, Message Result, Source ID*				*Data*			

Table 6.90: E2E Profile 8m example short - normal response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_OK)

6.19 Specification of E2E Profile 44m

[PRS_E2E_01155]

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈Profile 44m shall provide the following control fields, transmitted at runtime together with the protected data: Length, Counter, CRC, Data ID, Source ID, Message Type, Message Result (see [PRS_E2E_00910]).⌋

[PRS_E2E_00910] E2E Profile 44m mechanisms

Upstream requirements: RS_E2E_08529, RS_E2E_08530, RS_E2E_08533

⌈

Control field	Description
Length	16 bits, to support dynamic-size data.
Counter	16 bits.
CRC	32 bits, polynomial in normal form 0x1F4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCP/IP.
Data ID	32 bits, unique system-wide.
Message Type	2-bits, request (0) vs. response (1).
Message Result	2-bits, OK (0) vs. error (1) - fixed to OK (0) for Message Type 'request'.
Source ID	28-bits, unique system-wide.

⌋

For details of CRC calculation, the usage of start values and XOR values see SWS_CRCLibrary[3].

The specification of the profile uses the following placeholders:

Placeholder	Replacement
P44MLENGTH_POS	0
P44MLENGTH_LEN	2
P44MCOUNTER_POS	2
P44MCOUNTER_LEN	2
P44MDATAID_POS	4
P44MDATAID_LEN	4
P44MCRC_POS	8
P44MCRC_LEN	4
P44MMESSAGE_POS	12
P44MCALCULATE_CRC	Crc_CalculateCRC32P4()

Table 6.91: Profile 44m-specific data

For behavior and flowcharts of E2E method Profile 44m see Chapter 6.4.

6.19.1 Header Layout

In the E2E Profile 44m, the user data layout (of the data to be protected) is not constrained by E2E Profile 44m - there is only a requirement that the length of data to be protected is multiple of 1 byte.

The header of the E2E Profile 44m has one fixed layout, as follows:

Byte/Bit	0								1								2								3							
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	E2E Length																E2E Counter															
4	E2E Data ID																															
8	E2E CRC																															
12	T		R		E2E Source ID																											

Table 6.92: E2E Profile 44m Header

Hereby ‘T’ denotes the E2E Message Type and ‘R’ denotes the E2E Message Result. The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte first) - imposed by profile
2. LSB First (least significant bit within byte first) - imposed by TCP/IP network

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

6.19.2 Profile 44m Configuration Type

[PRS_E2E_01202]

Upstream requirements: RS_E2E_08528

⌈The E2E_P44mProtect, E2E_P44mForward, and E2E_P44mCheck functions "Config" shall have the following members defined in [PRS_E2E_00911].⌋

[PRS_E2E_00911] E2E Profile 44m Configuration Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
DataID	Unsigned Integer	A system-unique identifier of the Data.
Offset	Unsigned Integer	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 (16 bit) is written to Byte 1, the low Byte is written to Byte 2.
MinDataLength	Unsigned Integer	Minimal length of Data, in bits. E2E checks that DataLength is >= MinDataLength. The value shall be <= MaxDataLength and it shall be >= 16*8. The value shall be a multiple of 8.
MaxDataLength	Unsigned Integer	Maximal length of Data, in bits. E2E checks that DataLength is <= MaxDataLength. The value shall be <= 65535*8 and it shall be >= MinDataLength. The value shall be a multiple of 8.
MaxDeltaCounter	Unsigned Integer	Maximum allowed gap between two counter values of two consecutively received valid Data.

⌋

6.19.3 E2E Profile 44m Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P44mConfigType field	Value
DataID	0x0a0b0c0d
Offset	0x0000
MinDataLength	128
MaxDataLength	32768
MaxDeltaCounter	1

Table 6.93: E2E Profile 44m protocol example configuration

E2E_P44mProtectStateType field	Value
Counter	0

Table 6.94: E2E Profile 44m example state initialization

6.19.4 Request Example

The result data of an E2E_P44mProtect() call with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_REQUEST, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.95.

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x14	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	8	9	10	11	12	13	14	15
Data	0xae	0x67	0x4c	0xa0	0x00	0x12	0x34	0x56
Field	*CRC*				*Message Type, Message Result, Source ID*			
Byte	16	17	18	19	n/a			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.95: E2E Profile 44m example short - request (Message Type = STD_MESSAGETYPE_REQUEST; Message Result = STD_MESSAGERESULT_OK)

6.19.5 Response Example

The result data of an E2E_P44mProtect() call with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_OK is depicted in Table 6.96.

Byte	0	1	2	3	4	5	6	7
Data	0x00	0x14	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	8	9	10	11	12	13	14	15
Data	0x85	0x25	0x76	0x19	0x40	0x12	0x34	0x56
Field	*CRC*				*Message Type, Message Result, Source ID*			
Byte	16	17	18	19	n/a			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.96: E2E Profile 44m example short - normal response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_OK)

6.19.6 Error Response Example

The result data of an E2E_P44mProtect() call with short data length (length 20 bytes, means 4 actual data bytes), offset = 0, counter = 0, Source ID = 0x0123456, Message Type = STD_MESSAGETYPE_RESPONSE, Message Result = STD_MESSAGERESULT_ERROR is depicted in Table 6.97.

Byte	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data	0x00	0x14	0x00	0x00	0x0a	0x0b	0x0c	0x0d
Field	*Length*		*Counter*		*DataID*			
Byte	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Data	0x23	0x45	0x57	0x0f	0x50	0x12	0x34	0x56
Field	*CRC*				*Message Type, Message Result, Source ID*			
Byte	**16**	**17**	**18**	**19**	**n/a**			
Data	0x00	0x00	0x00	0x00	n/a			
Field	*Data*				*n/a*			

Table 6.97: E2E Profile 44m example short - ERROR response (Message Type = STD_MESSAGETYPE_RESPONSE; Message Result = STD_MESSAGERESULT_ERROR)

6.20 Specification of E2E state machine

The E2E Profile check()-function verifies data received in one cycle. This function only determines if data in that cycle are correct or not. In contrary, the state machine builds up a state out of several results of check() function within a reception window, which is then provided to the consuming application via middleware interfaces (e.g. RTE API).

The E2E state machine is applicable for all E2E profiles. It can be configured via parameters (see [PRS_E2E_00923]). The use of the E2E state machine is enabled/disabled by the disableEndToEndStateMachine parameter.

The configuration parameter "disableEndToEndStateMachine" determines if the state machine is used. If this parameter is set to TRUE then no state machine shall be used. In this case the application shall only receive the result of the latest E2E check but not any state information.

6.20.1 Overview of the state machine

This chapter gives an overview of the state machine function and contains a simplified representation of the behavior of the state machine. The states and transitions are included in full, but the transition conditions are generalized for better understanding.

The state machine provides the summarized result about the state of the communication channel by the following state information - E2E_SM_DEINIT, E2E_SM_INIT, E2E_SM_NODATA, E2E_SM_VALID, E2E_SM_INVALID (see [PRS_E2E_00596]).

The state transition depends on the current ProfileStatus of the E2E Profile check() and the ProfileStatus values of previous checks.

The state machine can be adjusted with the help of the parameter Config->transitToInvalidExtended.

[PRS_E2E_00675]

Upstream requirements: RS_E2E_08539

⌈By Config->transitToInvalidExtended==0 (false): The state machine shall behave according to Figure 6.131. There is no direct transition from E2E_SM_NODATA to E2E_SM_INVALID, no transition from E2E_SM_INIT to E2E_SM_INVALID due to counter-related faults (Autosar R19-11 or former behavior).⌋

[PRS_E2E_00676]

Upstream requirements: RS_E2E_08539

⌈By Config->transitToInvalidExtended==1 (true): The state machine shall behave according to Figure 6.132. The direct transition from E2E_SM_NODATA to E2E_SM_INVALID is covered, transition from E2E_SM_INIT to E2E_SM_INVALID due to counter-related faults is covered (state machine extended).⌋

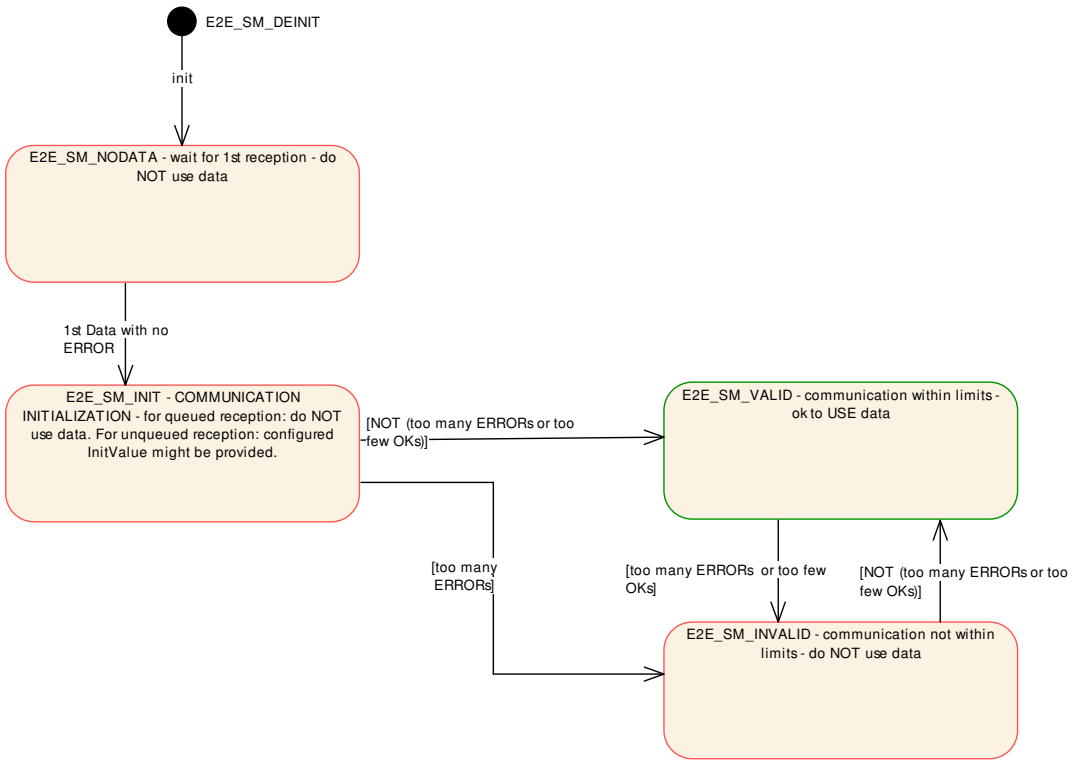

Figure 6.131: E2E state machine overview - [Config->transitToInvalidExtended == 0]

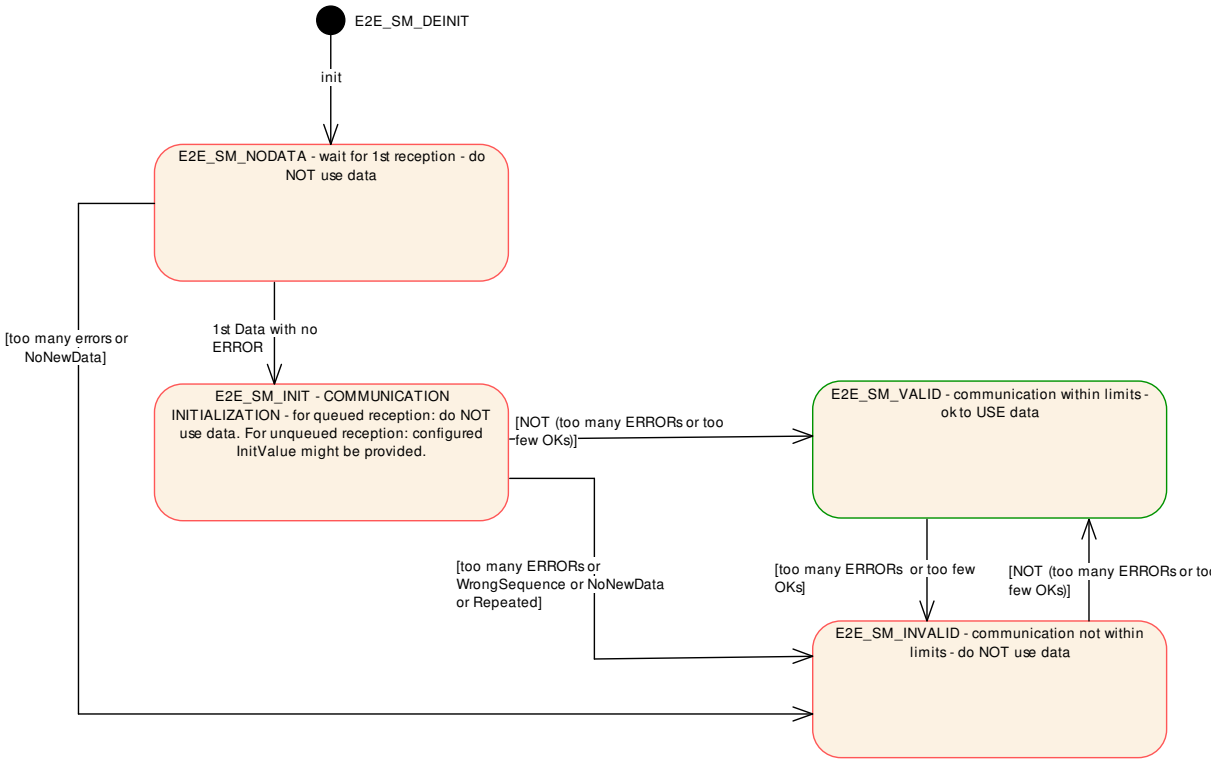

Figure 6.132: E2E state machine overview - [Config->transitToInvalidExtended == 1]

6.20.2 State machine specification

This chapter contains the detailed description of the behavior of the state machine. All transition conditions are described in full.

[PRS_E2E_00354]

Upstream requirements: RS_E2E_08539

⌈The E2E state machine shall be implemented by the functions E2E_SMCheck() and E2E_SMCheckInit()⌋

[PRS_E2E_00345]

Upstream requirements: RS_E2E_08539

⌈Depending on the configuration parameter Config->transitToInvalidExtended, the E2E State machine shall exhibit the behavior with respect to the function E2E_SMCheck() as shown in Figure 6.133 or Figure 6.134:

1. In case of Config->transitToInvalidExtended==0 (false): The E2E_SMCheck() shall behave according to Figure 6.133.
2. In case of Config->transitToInvalidExtended==1 (true): The E2E_SMCheck() shall behave according to Figure 6.134.
3. The current state (e.g. E2E_SM_VALID) is stored in State->SMState.

4. At every invocation of E2E_SMCheck, the ProfileStatus can be processed (as shown by logical step E2E_SMAddStatus()).
5. After that, there is an examination of three counters: State->ErrorCount and State->OKCount (continuously used through all different states) and State->NoDataInitCount (used in E2E_SM_NODATA and E2E_SM_INIT only). Depending on their values, there is a transition to a new state, stored in State->SMState.
6. The transitioning to another state triggers a reset of some values by E2E_SMClearStatus() (see [PRS_E2E_00467]) or E2E_SMClearRemainingStatus() (see [PRS_E2E_00607]).

⌋

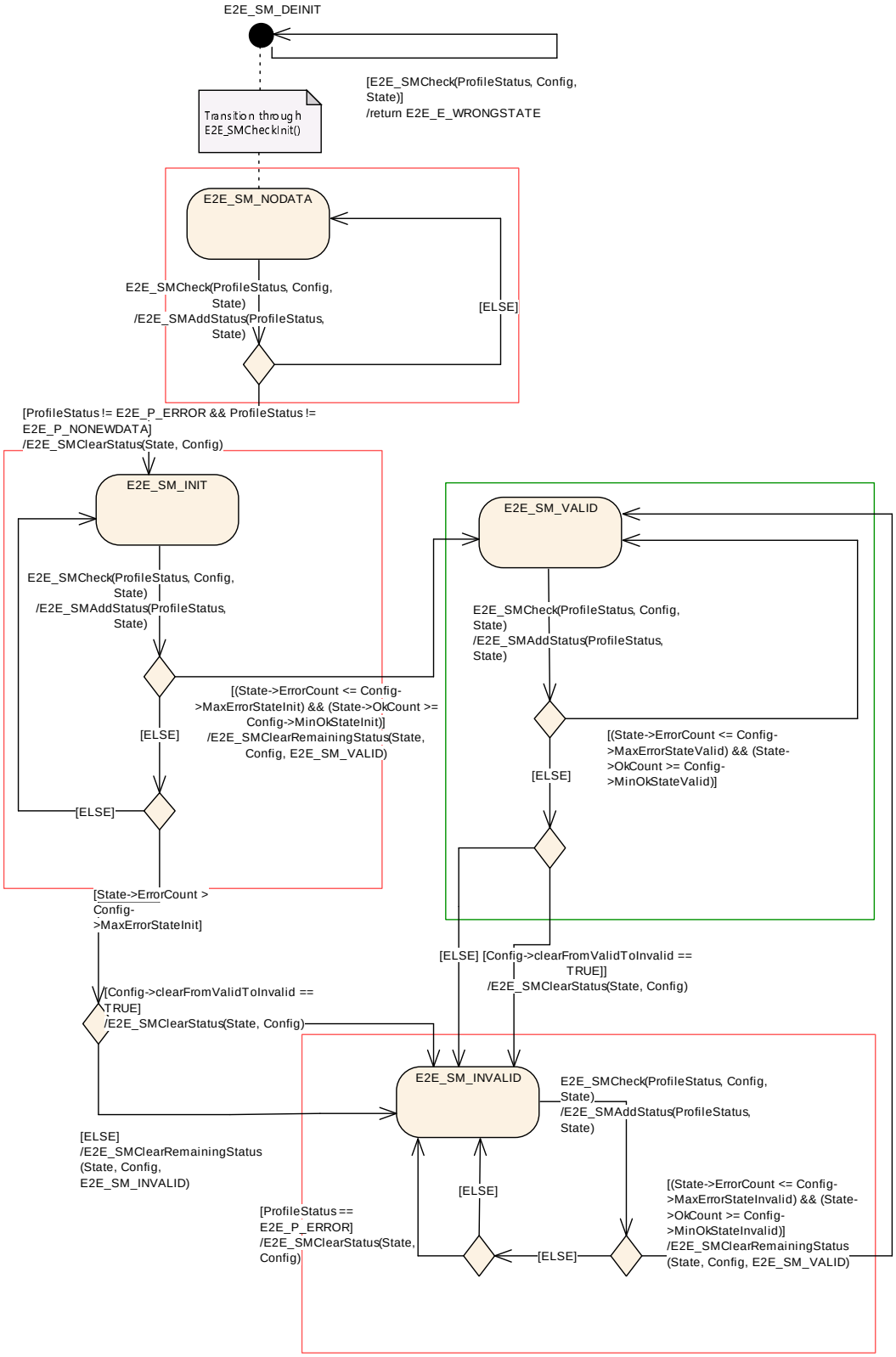

Figure 6.133: E2E state machine check - [Config->transitToInvalidExtended == 0]

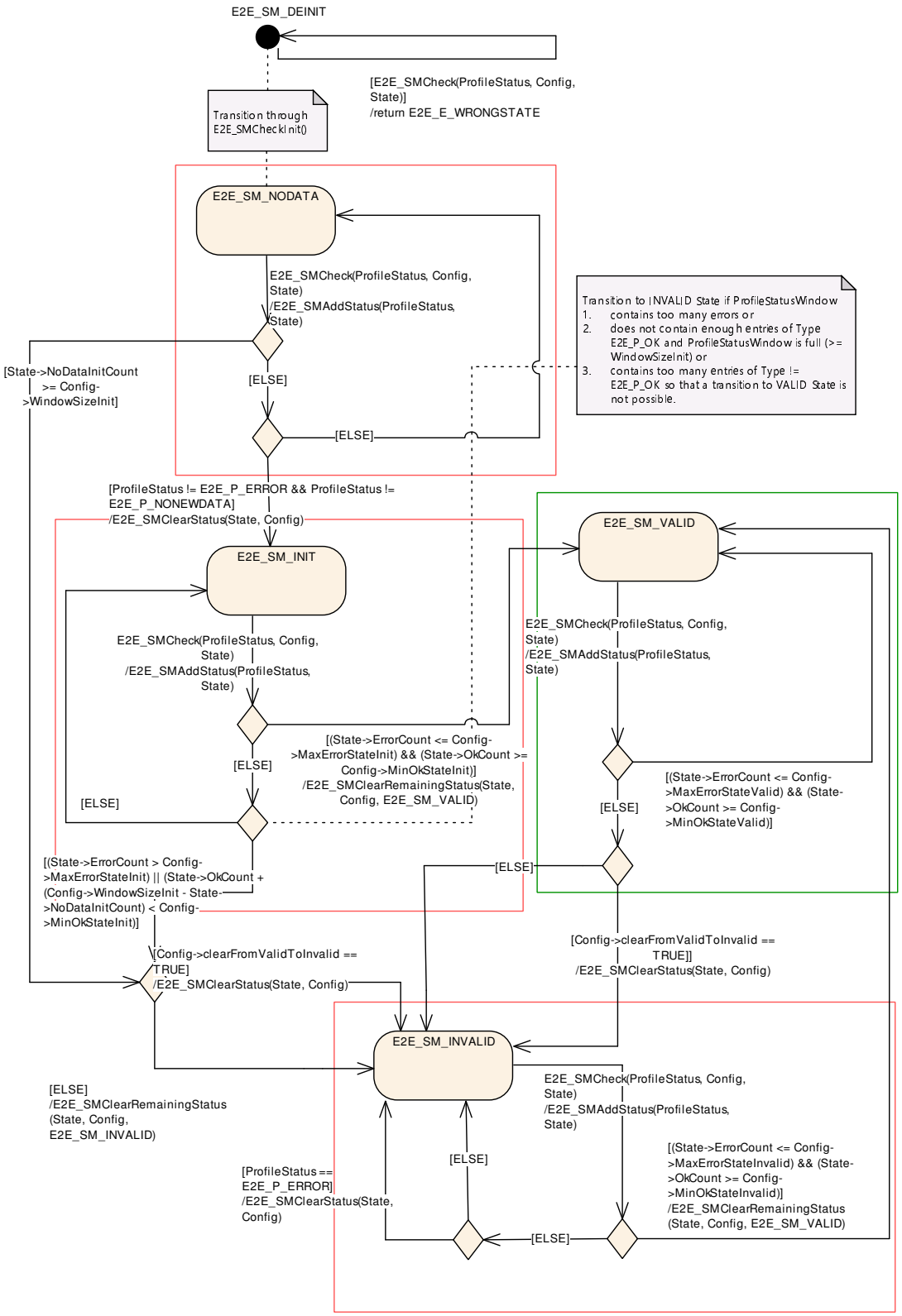

Figure 6.134: E2E state machine check - [Config->transitToInvalidExtended == 1]

The conditions for the transitions between the states are included in the graphical representation in Figure 6.133 and Figure 6.134. However the following textual explanation of the state transitions is added for better understanding.

Any transition is triggered based on the result of the E2E check function and the current E2E state machine state. The result of the E2E check function can be summarized as one of the following:

- valid data - the E2E check gives the value E2E_P_OK.
- corrupted data - the E2E check gives the value E2E_P_ERROR.
- no new data - the E2E check gives the value E2E_P_NONEWDATA.

Most transitions are independent of the value set for Config->transitToInvalidExtended. The transitions that are valid only for a specific setting of Config->transitToInvalidExtended are marked at the end with (if Config->transitToInvalidExtended==1) or (if Config->transitToInvalidExtended==0).

6.20.2.1 Transition from E2E_SM_NODATA

The transition from E2E_SM_NODATA to E2E_SM_INIT is triggered whenever the latest received data is no corrupted data from E2E point of view.

The State->NoDataInitCount sums up count of invalid data in both states E2E_SM_NODATA and E2E_SM_INIT. The following behavior in E2E_SM_INIT can be adjusted with the help of the parameter Config->combinedNoDataInitCount (see [PRS_E2E_01437]).

In case of Config->combinedNoDataInitCount == 0 (false): the value of State->NoDataInitCount will be reset while transitioning from E2E_SM_NODATA to E2E_SM_INIT. Therefore, State->NoDataInitCount will not change the behavior of the statemachine compared to R23-11. In case of Config->combinedNoDataInitCount == 1 (true): the value of State->NoDataInitCount will not be reset while transitioning from E2E_SM_NODATA to E2E_SM_INIT. Therefore, State->NoDataInitCount will change the behavior of the statemachine compared to R23-11. Please see Chapter 6.20.4 (FTTI and E2E parameters) for more information.

The transition from E2E_SM_NODATA to E2E_SM_INVALID is triggered whenever within an interval of WindowSizeInit message cycles either no new data or corrupted data or a mixture of both have been detected. (if Config->transitToInvalidExtended==1)

6.20.2.2 Transition from E2E_SM_INIT

The transition from E2E_SM_INIT to E2E_SM_VALID is triggered whenever within an interval of WindowSizeInit message cycles less than the number of configured corrupted data and more or equal than the number of configured valid data have been detected.
The transition from E2E_SM_INIT to E2E_SM_INVALID is triggered whenever within an interval of WindowSizeInit message cycles more than the configured number of corrupted data have been detected. (if Config->transitToInvalidExtended==0)

The transition from E2E_SM_INIT to E2E_SM_INVALID is triggered whenever within an interval of WindowSizeInit message cycles more than the configured number of corrupted data or less than the configured number of valid data have been detected. (if Config->transitToInvalidExtended==1) This behavior can be adjusted with the help of the parameter Config->combinedNoDataInitCount (see [PRS_E2E_01437]). In case of Config->combinedNoDataInitCount ==1 (true): The State->NoDataInitCount sums up the total count of invalid data in both states E2E_SM_NODATA and E2E_SM_INIT. Therefore, the value of State->NoDataInitCount will not be reset while transitioning from E2E_SM_NODATA to E2E_SM_INIT and can reach the threshold of the configured number of corrupted data earlier then in statemachine version of R23-11. Please see Chapter 6.20.4 (FTTI and E2E parameters) for more information.

6.20.2.3 Transition from E2E_SM_VALID

The transition from E2E_SM_VALID to E2E_SM_INVALID is triggered whenever within an interval of WindowSizeValid message cycles more than the configured number of corrupted data or less than the configured number of valid data have been detected.

6.20.2.4 Transition from E2E_SM_INVALID

The transition from E2E_SM_INVALID to E2E_SM_VALID is triggered whenever within an interval of WindowSizeInvalid message cycles less than the number of configured corrupted data and more than the number of configured valid data have been detected.

A number of functions are used in the state machine:

- E2E_SMAddStatus()
- E2E_SMCheckInit()
- E2E_SMClearStatus()
- E2E_SMClearRemainingStatus()

E2E_SMAddStatus is just a logical step in the algorithm, it may (but does not have to be) implemented as a separate function. It is not a module API function. The value State->OKCount represents the number of received E2E_P_OK status. The value State->Error Counter represents the number of E2E_P_ERROR status. The remaining status values represent counter errors, which doo not contribute to State->OKCount ir State->ErrorCount.

[PRS_E2E_00466]

Upstream requirements: RS_E2E_08539

⌈The step E2E_SMAddStatus(ProfileStatus, State) in E2E_SMCheck() shall behave as shown in Figure 6.135.⌋

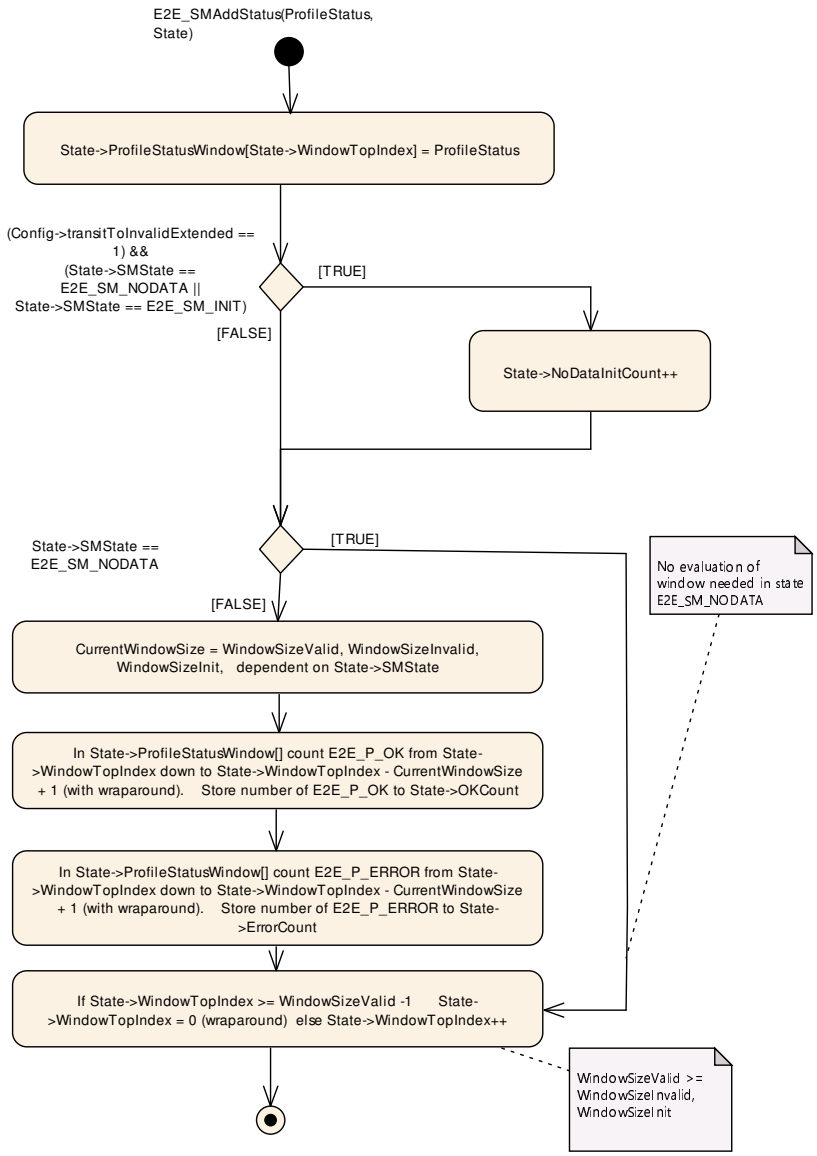

Figure 6.135: E2E state machine step E2E_SMAddStatus

E2E_SMCheckInit() is applied to initialize the state machine based on the selected configuration.

[PRS_E2E_00375]

Upstream requirements: RS_E2E_08539

⌈The E2E State machine shall have the behavior with respect to the function E2E_SMCheckInit() as shown in Figure 6.136.⌋

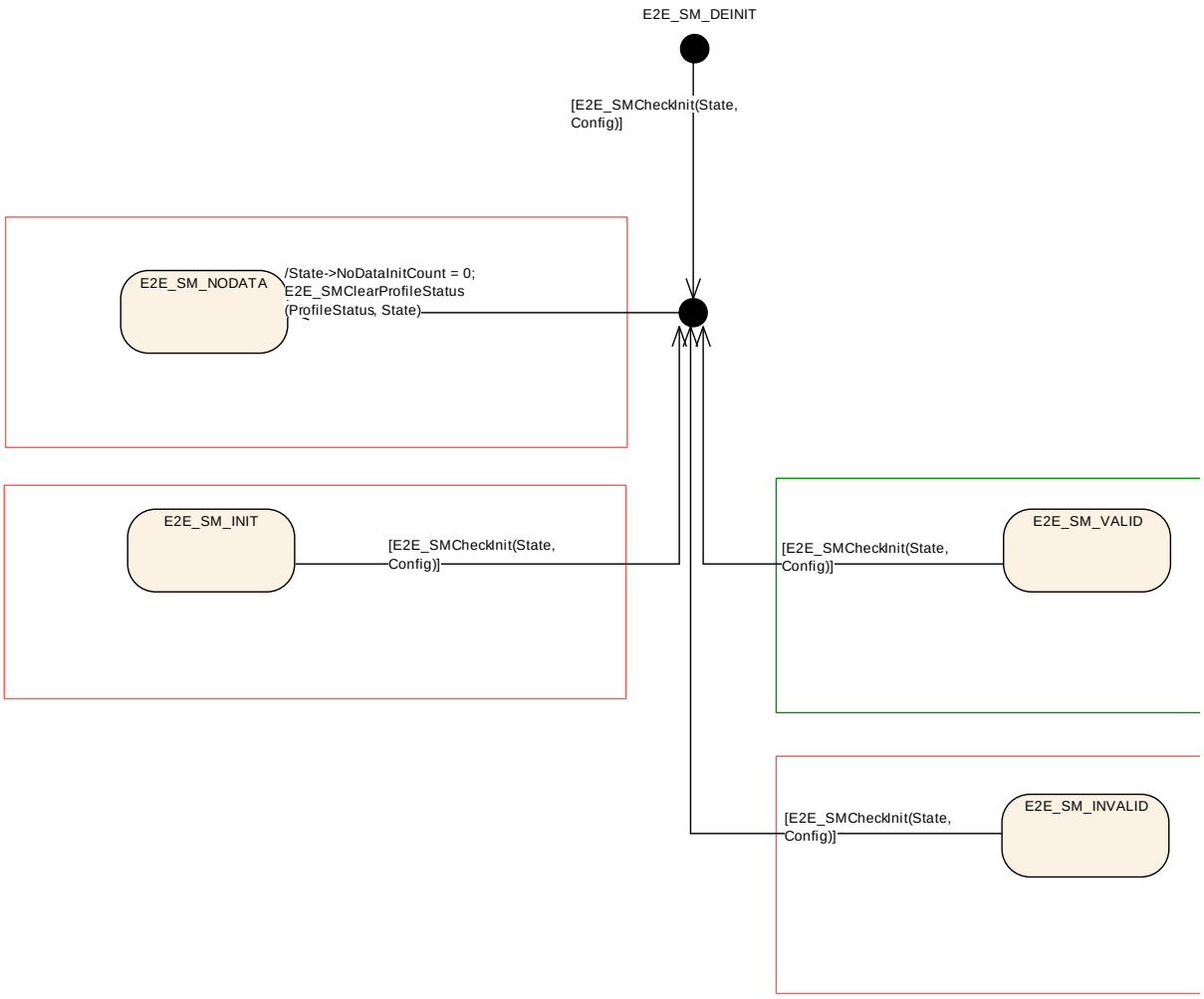

Figure 6.136: E2E state machine step E2E_SMCheckInit

E2E_SMClearStatus() initializes the whole monitoring window (ProfileStatusWindow) to start with a cleared window. This clearing is done by setting all entries of ProfileStatusWindow to value E2E_P_NOTAVAILABLE.

[PRS_E2E_00467]

Upstream requirements: RS_E2E_08539

⌈The step E2E_SMClearStatus(State, Config) in E2E_SMCheck() shall behave as shown in Figure 6.137.⌋

[PRS_E2E_01437] Parameter Config->combinedNoDataInitCount

Upstream requirements: RS_E2E_08539

⌈Depending on the configuration parameter Config->combinedNoDataInitCount, the E2E_SMClearStatus shall exhibit the behavior with respect to the function E2E_SMCheck() as shown in Figure 6.137.⌋

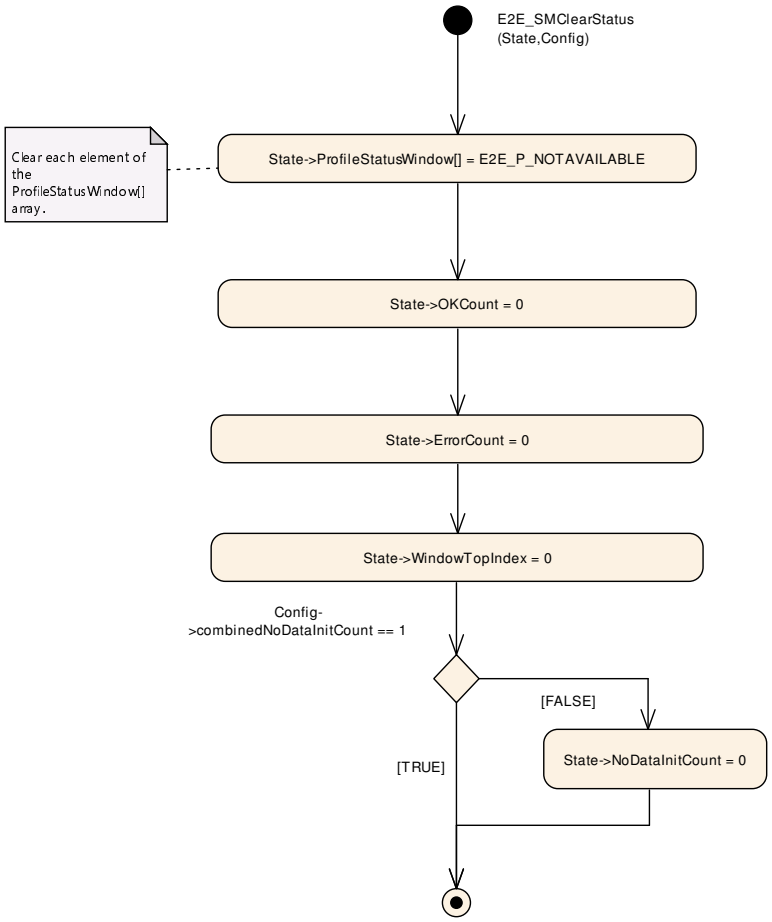

Figure 6.137: E2E state machine step E2E_SMClearStatus

E2E_SMClearRemainingStatus() initializes a part of the monitoring window (ProfileStatusWindow) if the WindowSize increases due to a state transition. In this case the additional Window entries at the beginning are cleared by setting them to E2E_P_NOTAVAILABLE.

[PRS_E2E_00607]

Upstream requirements: RS_E2E_08539

⌈The step E2E_SMClearRemainingStatus(Config, State, NextState) in E2E_SMCheck() shall have the following behavior: Figure 6.138.⌋

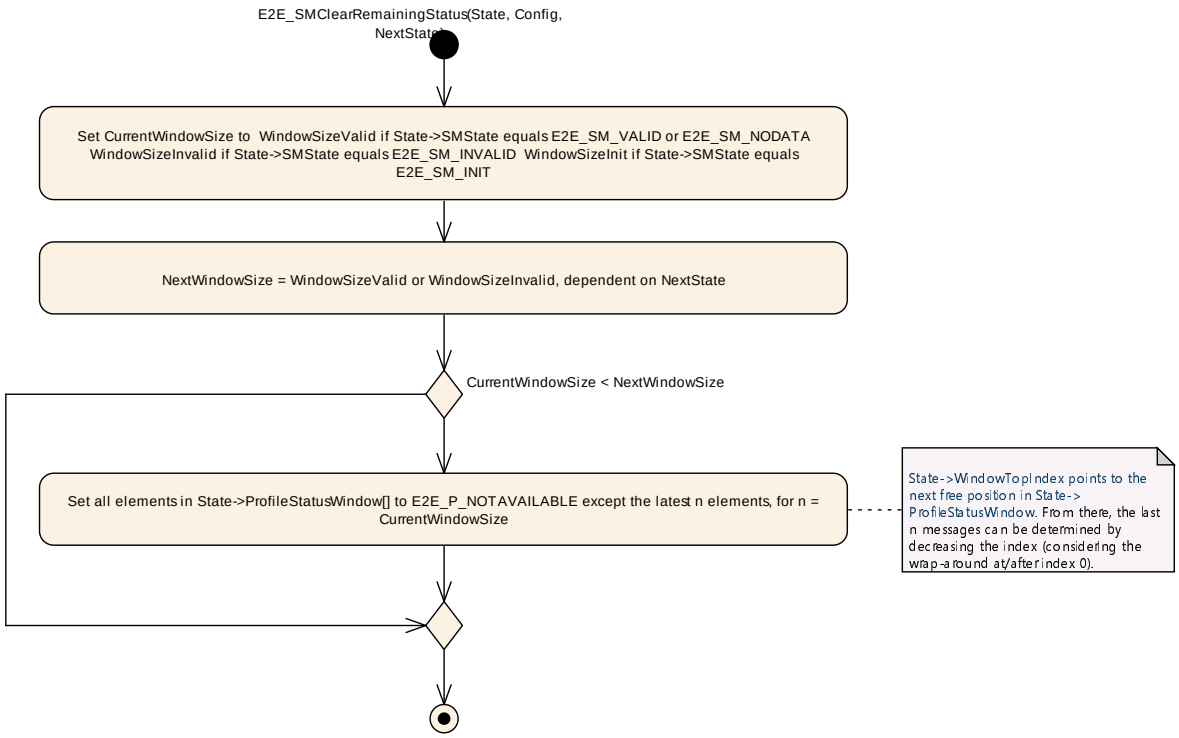

Figure 6.138: E2E state machine step E2E_SMClearRemainingStatus

E2E_SMClearStatus() and E2E_SMClearRemainingStatus() are both applied if a transition to E2E_SM_INVALID is triggered. The parameter Config–>clearFromValidToInvalid indicates which function is to be applied:

Config->clearFromValidToInvalid = True: E2E_SMClearStatus()
Config->clearFromValidToInvalid = False: E2E_SMClearRemainingStatus()

6.20.3 State Machine Types

6.20.3.1 E2E State Machine Configuration Type

[PRS_E2E_00668]

Upstream requirements: RS_E2E_08528

⌈The E2E State Machines 'Config' object shall have the members as defined in [PRS_E2E_00912].⌋

[PRS_E2E_00912] E2E State Machine Configuration Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
WindowSizeValid	Unsigned Integer	Size of the monitoring window for the state machine during state VALID.

WindowSizeInit	Unsigned Integer	Size of the monitoring windows for the state machine during state INIT.
WindowSizeInvalid	Unsigned Integer	Size of the monitoring window for the state machine during state INVALID.
MaxErrorStateInit	Unsigned Integer	Maximal number of checks in which ProfileStatus equals to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INIT).
MinOkStateValid	Unsigned Integer	Minimal number of checks in which ProfileStatus equals 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	Unsigned Integer	Maximal number of checks in which ProfileStatus equals to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_VALID).
MinOkStateInvalid	Unsigned Integer	Minimum number of checks in which ProfileStatus equals 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	Unsigned Integer	Maximal number of checks in which ProfileStatus equals to E2E_P_ERROR was determined, within the last WindowSize checks (for the state E2E_SM_INVALID).
clearFromValidToInvalid	Boolean	Clear monitoring window data on transition to state INVALID.
transitToInvalidExtended	Boolean	E2E State machine behavior concerning transition E2E_SM_NODATA or E2E_SM_INIT to E2E_SM_INVALID.
combinedNoDataInitCount	Boolean	E2E State machine behavior concerning transition E2E_SM_NODATA or E2E_SM_INIT to E2E_SM_INVALID.

⌋

6.20.3.2 E2E State Machine State Type

[PRS_E2E_00669] E2E State Machine State Type

Upstream requirements: RS_E2E_08528

⌈The E2E State Machines 'State' object shall have the members defined in [PRS_E2E_00913].⌋

[PRS_E2E_00913] E2E State Machine State Type

Upstream requirements: RS_E2E_08528

⌈

Name	Type	Description
ProfileStatusWindow	Unsigned Integer Array	An array in which the ProfileStatus values of the last E2E-checks are stored. The array size shall be WindowSizeValid. This array holds the results of the E2E checks for latest WindowsSize (see table 'E2E State Machine Check Status Enumeration'). To initialize the elements of the array the value E2E_P_NOTAVAILABLE is used. Therefore the whole or parts of the array are to be initialized with E2E_P_NOTAVAILABLE (see E2E_SMClearStatus, E2E_SMClearRemainingStatus) The values to be stored in the array are the results of the latest E2E check (see table 'E2E State Machine Check Status Enumeration') and the initialization value E2E_P_NOTAVAILABLE. If no results of E2E checks shall be stored (E2E_SMClearStatus, E2E_SMClearRemainingStatus) entries shall be initialized with value E2E_P_NOTAVAILABLE.
WindowTopIndex	Unsigned Integer	index in the array, at which the next ProfileStatus is to be written.
OkCount	Unsigned Integer	Count of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks.
ErrorCount	Unsigned Integer	Count of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks.
NoDataInitCount	Unsigned Integer	Count of checks in the state E2E_SM_NODATA or E2E_SM_INIT without transitioning to the state E2E_SM_VALID.
SMState	Enumeration	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.

⌋

6.20.3.3 E2E State Machine Status Enumeration

[PRS_E2E_00596]

Upstream requirements: RS_E2E_08528

⌈The E2E State Machine uses the following enumeration values to indicate its current status as defined in [PRS_E2E_00914]).⌋

[PRS_E2E_00914] E2E State Machine Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	Description
E2E_SM_VALID	Communication functioning properly according to E2E, data can be used.
E2E_SM_DEINIT	State before E2E_SMCheckInit() is invoked, data cannot be used.
E2E_SM_NODATA	No data from the sender is available since the initialization, data cannot be used.
E2E_SM_INIT	There has been some data received since startup, but it is not yet possible use it, data cannot be used.
E2E_SM_INVALID	Communication not functioning properly, data cannot be used.

⌋

6.20.3.4 Profile specific Check Status to State Machine Check Status Mappings

This section targets the single mappings between each Profile specific check state to the check states used by the E2E State Machine.

[PRS_E2E_00597]

Upstream requirements: RS_E2E_08528

⌈The E2E State Machine uses the following enumeration values as input from the Profile specific check functions as defined in [PRS_E2E_00915]).⌋

[PRS_E2E_00915] E2E State Machine Check Status Enumeration

Upstream requirements: RS_E2E_08528

⌈

Name	Description
E2E_P_OK	Check of the message was successful and no error was found
E2E_P_ERROR	An error was detected in the received message.
E2E_P_REPEATED	A repeated counter was received
E2E_P_NONEWDATA	No new message was received
E2E_P_WRONGSEQUENCE	The received message contains wrong counter

⌋

[PRS_E2E_00598] Mapping Profile 1 to State Machine

Upstream requirements: RS_E2E_08528

⌈The mapping between Profile 1 specific check states to the input for the E2E State Machine since R4.2 is described in [PRS_E2E_00916]).⌋

[PRS_E2E_00916] E2E Profile 1 specific Check Status Mapping since R4.2

Upstream requirements: RS_E2E_08528

⌈

Profile Specific State	State Machine State
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

⌋

[PRS_E2E_00641] Mapping Profile 1 to State Machine

Upstream requirements: RS_E2E_08528

⌈The mapping between Profile 1 specific check states to the input for the E2E State Machine prior to R4.2 is described in [PRS_E2E_00917].⌋

[PRS_E2E_00917] E2E Profile 1 specific Check Status Mapping prior to R4.2

Upstream requirements: RS_E2E_08528

⌈

Profile Specific State	State Machine State
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

⌋

[PRS_E2E_00599] Mapping Profile 2 to State Machine

Upstream requirements: RS_E2E_08528

⌈The mapping between Profile 2 specific check states to the input for the E2E State Machine since R4.2 is described in [PRS_E2E_00918]).⌋

[PRS_E2E_00918] E2E Profile 2 specific Check Status Mapping since R4.2

Upstream requirements: RS_E2E_08528

⌈

Profile Specific State	State Machine State
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

⌋

[PRS_E2E_00670] Mapping Profile 2 to State Machine

Upstream requirements: RS_E2E_08528

⌈The mapping between Profile 2 specific check states to the input for the E2E State Machine prior to R4.2 is described in [PRS_E2E_00919].⌋

[PRS_E2E_00919] E2E Profile 2 specific Check Status Mapping prior to R4.2

Upstream requirements: RS_E2E_08528

⌈

Profile Specific State	State Machine State
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

⌋

[PRS_E2E_00600] Mapping Profile 4,5,6,7,8,11,22,44,4m,7m,8m,44m to State Machine

Upstream requirements: RS_E2E_08528

⌈The mapping between Profile 4 specific check states to the input for the E2E State Machine is described in [PRS_E2E_00920] (yy is one of the profiles 04,05,06,07,08,11,22,44,04m,07m,08m,44m).⌋

[PRS_E2E_00920] E2E Profile specific Check Status Mapping

Upstream requirements: RS_E2E_08528

⌈

Profile Specific State	State Machine State
E2E_PyySTATUS_OK, E2E_PyySTATUS_OKSOMELOST	E2E_P_OK
E2E_PyySTATUS_ERROR	E2E_P_ERROR
E2E_PyySTATUS_REPEATED	E2E_P_REPEATED
E2E_PyySTATUS_NONEWDATA	E2E_P_NONEWDATA
E2E_PyySTATUS_WRONGSEQUENCE	E2E_P_WRONGSEQUENCE

⌋

6.20.4 FTTI and E2E Parameters

Considering the suggestion, switching to safe state by INVALID, the E2E state machine parameters can be chosen in such a way that the FTTI of the E2E fault can be fulfilled. Usually, FTTI =tFD+tFR, where tFD is fault detection time (E2E fault-debouncing time), and tFR is fault reaction time (time for SW/HW fault reaction to switch to safe state after the fault detection). By neglectable tFR, FTTI=tFD.
The following hints can be considered by determining the E2E parameters:

1. WindowSizeInit-1 <= tFD/cycleTime. For example, by state E2E_SM_NODATA, the state machine will switch to E2E_SM_INVALID if the number of NoNewData reaches WindowSizeInit, and the duration is (WindowSizeInit-1)*CycleTime. By state E2E_SM_INIT, the fault detection time is then (WindowSizeInit-MinOkStateInit)*CycleTime.
   In case of Config->combinedNoDataInitCount ==0(false): By state transition from E2E_SM_NODATA to E2E_SM_INIT, and then to E2E_SM_INVALID (e.g. first NoNewData, then WrongSequence), the total duration might exceed tFD. If this is relevant, then the following calculation can be used: 2 * WindowSizeInit-2 <= tFD/cycleTime.
2. WindowSizeValid-MinOkStateValid <= tFD/cycleTime. For example, by state E2E_SM_VALID, the state machine will switch to E2E_SM_INVALID, if the number of WrongSequence reaches (WindowSizeValid-MinOkStateValid+1), and the duration is (WindowSizeValidMinOkStateValid)*CycleTime.

7 E2E API specification

This chapter defines an abstract API of E2E supervision. E2E is supposed to be invoked by middleware, but the results of checks are visible to the application, so this chapter is split into two parts.

7.1 API of middleware to applications

The API to the applications is imposed by the middleware (e.g. RTE or ara::com). E2E provides an additional output object providing E2E check results.

[PRS_E2E_01443] Interface to middleware

Replaces: PRS_E2E_UC_00321
Upstream requirements: RS_E2E_08534

⌈The middleware should provide, for each exchanged dataRecord, a set of functions:

- middleware_send(in dataRecord)
- middleware_receive(out dataRecord, out e2eResult)

⌋

[PRS_E2E_00322]

Upstream requirements: RS_E2E_08534

⌈The e2eResult shall contain information:

- e2eStatus: Profile-independent status of the reception on one single Data in one cycle. Possible values are: OK, REPEATED, WRONGCRC, WRONGSEQUENCE, NONEWDATA.

⌋

[PRS_E2E_00853]

Upstream requirements: RS_E2E_08534

⌈The e2eResult shall contain information:

- e2eState: Status of the communication channel exchanging the data. Possible values are: VALID, DEINIT, NODATA, INIT, INVALID if the state machine is enabled (disableEndToEndStateMachine = FALSE).

If the state machine is disabled then e2eResult shall not contain a status information of the communication channel.⌋

[PRS_E2E_00677]

Upstream requirements: RS_E2E_08534, RS_E2E_08528

⌈The mapping between profile independent and profile specific states shall be as described in [PRS_E2E_00921]): (yy is one of the profiles 01,02,04,05,06,07,08,11,22,44,04m,07m,08m,44m)⌋

[PRS_E2E_00921] Mapping between profile independent and specific states

Upstream requirements: RS_E2E_08534, RS_E2E_08528

⌈

Profile independent result	Profile specific result with yy one of profiles (01,02,04,05,06,07,08,11,22,44,04m,07m,08m,44m)
OK	E2E_PyySTATUS_OK
OK	E2E_PyySTATUS_OKSOMELOST
WRONGCRC	E2E_PyySTATUS_ERROR
REPEATED	E2E_PyySTATUS_REPEATED
NONEWDATA	E2E_PyySTATUS_NONEWDATA
WRONGSEQUENCE	E2E_PyySTATUS_WRONGSEQUENCE

⌋

[PRS_E2E_00678]

Upstream requirements: RS_E2E_08534, RS_E2E_08548

⌈The mapping between communication channel status and state machine states shall be as defined in [PRS_E2E_00922].⌋

[PRS_E2E_00922] Mapping between Communication Channel and State Machine States

Upstream requirements: RS_E2E_08534, RS_E2E_08548

⌈

Communication channel status	Result of E2E_SMCheck
VALID	E2E_SM_VALID
NODATA	E2E_SM_NODATA
DEINIT	E2E_SM_DEINIT
INIT	E2E_SM_INIT
INVALID	E2E_SM_INVALID

⌋

7.2 API of E2E

The E2E interface is independent from any middleware. It is designed with SOME/IP in mind, but it could work for any other middleware or software services, e.g. a database requesting to protect its data.

The interface between the middleware and E2E operates on the serialized data, where: E2E adds E2E header (sender side) and E2E check E2E header (receiver side).

[PRS_E2E_00323]

Upstream requirements: RS_E2E_08534

⌈E2E protocol shall provide the following interface:

- E2E_check(in dataID, inout serializedData): e2eResult
- E2E_protect(in dataID, inout serializedData)
- E2E_forward(in dataID, inout serializedData)

where:

- dataID is a unique identifier of the exchanged data/information. In case of multiple instantiation, each single instance gets typically a separate dataID, even if the same type of information is transmitted
- serializedData - vector/array of serialized data, where E2E header is located, next to serialized data
- e2eResult - result of E2E checks, see previous section for the definition.

⌋

[PRS_E2E_00828]

Upstream requirements: RS_E2E_08534

⌈For C/S (method) communication the E2E protocol shall provide the following interface:

- E2E_check(in dataID, in messageType, in messageResult, in sourceID, in serializedData): e2eResult
- E2E_check(in dataID, in messageType, in messageResult, out sourceID, in serializedData): e2eResult
- E2E_protect(in dataID, in messageType, in messageResult, in sourceID, inout serializedData)
- E2E_forward(in dataID, in messageType, in messageResult, in sourceID, inout serializedData)

where:

- dataID is a unique identifier of the exchanged data/information. In case of multiple instantiation, each single instance gets typically a separate dataID, even if the same type of information is transmitted
- messageType is used to distinguish between request and response messages
- messageResult is used to distinguish between normal response and error response messages

- sourceID is a unique identifier of the source of the exchanged data/information. In case C/S (method) communication, each single client gets a separate sourceID.
- serializedData - vector/array of serialized data, where E2E header is located, next to serialized data
- e2eResult - result of E2E checks, see previous section for the definition.

Note that there are two overloads for E2E_check() that differ w.r.t. the direction of the sourceID. On the source side (i.e., the client side in case of C/S communication) the sourceID shall be provided by the caller to the E2E_check() in order to have the sourceID in the E2E header verified against the passed one. On the sink side (i.e., the server side in case of C/S communication) the sourceID shall be extracted from the E2E header by E2E_check() and shall be provided to caller.⌋

The middleware is responsible to provide an adaptation to E2E functional interface.

[PRS_E2E_00318]

Upstream requirements: RS_E2E_08534

⌈The middleware shall determine the DataID, the Message Type, the Message Result, and the SourceID of the currently exchanged information.⌋

For example, in case of SOME/IP events, it needs to determine DataID based on serviceid/eventid/instanceid tuple.

[PRS_E2E_00319]

Upstream requirements: RS_E2E_08534

⌈The middleware invokes E2E functions providing them the DataID the Message Type, the Message Result, and the SourceID together with the data.⌋

[PRS_E2E_00320]

Upstream requirements: RS_E2E_08534

⌈On the receiver side, the middleware shall provide the e2eResult determined by E2E to the receiver.⌋

8 Configuration Parameters

E2E supervision has the following configuration options for each protected data. Note that it is platform-specific how middleware associates a middleware communication channel with E2E communication protection.

For each DataID, which uniquely represents data exchanged, there is a set of configuration options.

[PRS_E2E_00324]

Upstream requirements: RS_E2E_08534, RS_E2E_08542

⌈The options for a E2E-protected data shall be available as defined in [PRS_E2E_00923], [PRS_E2E_00924] and [PRS_E2E_00925].⌋

[PRS_E2E_00923] Disable/enable configuration parameters

Upstream requirements: RS_E2E_08534, RS_E2E_08542

⌈

Parameters	Description
disableEndToEndCheck	Disables/Enables the E2E check. The E2Eheader is removed from the payload regardless from the setting of this attribute.
disableEndToEndStateMachine	Disables/Enables the E2E state machine. The E2E check is applied regardless from the setting of this attribute. In case disableEndToEndCheck is set, then disableEndToEndStateMachine should be set as well.

⌋

[PRS_E2E_00924] E2E profile specific configuration parameters

Upstream requirements: RS_E2E_08534, RS_E2E_08542

⌈

Parameters	Profile	Description
profileName	all	This represents the identification of the concrete E2E profile. Possible profiles: 1 (only CP), 2 (only CP), 4, 5, 6, 7, 8, 11, 22, 44, 4m, 7m, 8m, 44m.
dataID	1, 4, 5, 6, 7, 8, 11, 44, 4m, 7m, 8m, 44m	This represents a unique numerical identifier. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEnd-Protection. dataId is used as a unique identifier of a configuration object. One dataId can appear only once in the configuration.
sourceID	4m, 7m, 8m, 44m	This represents a unique numerical identifier. Note: ID is used for protection against masquerading among different sources producing the same data w.r.t. the dataID. sourceId is used as a unique identifier of the source producing certaing data. One sourceId can appear only once in the configuration.
dataLength	1, 2, 5, 11, 22	For fixed size data: length of data in bits.

▽

△

minDataLength	4, 6, 7, 8, 44, 4m, 7m, 8m, 44m	For variable size data: minimum length of data in bits.
maxDataLength	4, 6, 7, 8, 44, 4m, 7m, 8m, 44m	For variable size data: maximum length of data in bits.
dataIdList	2, 22	List of 16 dataID values, where a different value is transmitted depending on counter value.
dataIdMode	1, 11	This attribute describes the inclusion mode that is used to include the two-byte Data ID in E2E communication protection.
offset	2, 4, 5, 6, 7, 8, 22, 44, 4m, 7m, 8m, 44m	Offset of the E2E header in the Data[] array in bits.
counterOffset	1, 11	Offset of the counter in the Data[] array in bits. Fixed for AP to 8.
crcOffset	1, 11	Offset of the CRC in the Data[] array in bits. Fixed for AP to 0.
dataIdNibbleOffset	1, 11	Offset of the dataID nibble in the Data[] array in bits. Fixed for AP to 12.
maxDeltaCounter	4, 5, 6, 7, 8, 11, 22, 44, 4m, 7m, 8m, 44m	Maximum allowed difference between the counter value of the current message and the previous valid message.
Parameters of legacy profiles (Only CP)		
maxDeltaCounterInit	1, 2	Initial maximum allowed gap between two counter values of two consecutively received valid Data. The maxDeltaCounter is increased on each reception try but only reset when receiving a valid message. This is to compensate for and tolerate lost messages.
maxNoNewOrRepeated-Data	1, 2	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
syncCounterInit	1, 2	The number of messages required for validating the consistency of the counter after exceeding the maxNoNewOrRepeatedData threshold.
profileBehavior	1, 2	Mapping of specific profile status values to unified profileStatus. False: legacy behavior, as before AUTOSAR Classic Platform Release 4.2,True: mapping according to new profiles (profile 4 and newer) interpretation of status, introduced in AUTOSAR Classic Platform Release 4.2.

⌋

[PRS_E2E_00925] E2E state machine configuration parameters

Upstream requirements: RS_E2E_08534, RS_E2E_08542

⌈

Parameters	Description
transitToInvalidExtended	E2E State machine behavior concerning transition from E2E_SM_NODATA or E2E_SM_INIT to E2E_SM_INVALID Value=0 (false): no direct transition from E2E_SM_NODATA to E2E_SM_INVALID, no transition from E2E_SM_INIT to E2E_SM_INVALID due to counter-related faults (behavior as in Autosar R19-11 or before) value=1 (true): direct transition from E2E_SM_NODATA to E2E_SM_INVALID covered, transition from E2E_SM_INIT to E2E_SM_INVALID due to counter-related faults covered (state machine extended)

▽

△

combinedNoDataInitCount	E2E State machine behavior concerning transition from E2E_SM_NODATA or E2E_SM_INIT to E2E_SM_INVALID Value=0 (false) or not defined: Count of E2E checks in state E2E_SM_NODATA or E2E_SM_INIT without transitioning to the state E2E_SM_INVALID is separated per state. See Chapter 6.20.4 for hints with respect to FTTI. (Behavior as in AUTOSAR R23-11 or before). value=1 (true): Count of E2E checks in the state E2E_SM_NODATA or E2E_SM_INIT without transitioning to the state E2E_SM_INVALID is combined.
windowSizeValid	Size of the monitoring window of state Valid for the E2E state machine.
windowSizeInvalid	Size of the monitoring window of state Invalid for the E2E state machine.
windowSizeInit	Size of the monitoring window of state Init for the E2E state machine.
clearFromValidToInvalid	Clear monitoring window on transition from state Valid to state Invalid.
maxErrorStateInit	Maximum number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSizeInit checks, for the state E2E_SM_INIT.
maxErrorStateInvalid	Maximum number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSizeInvalid checks, for the state E2E_SM_INVALID.
maxErrorStateValid	Maximum number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSizeValid checks, for the state E2E_SM_VALID.
minOkStateInit	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSizeInit checks, for the state E2E_SM_INIT.
minOkStateInvalid	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSizeInvalid checks, for the state E2E_SM_INVALID.
minOkStateValid	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSizeValid checks, for the state E2E_SM_VALID.

⌋

8.1 General Constraints

This section contains general platform independent constraints. These belong to the configuration parameters mentioned in [PRS_E2E_00925].

8.1.1 E2E State Machine Settings

[PRS_E2E_01457] Value range of `windowSizeValid`

Replaces: PRS_E2E_CONSTR_03176
Upstream requirements: RS_E2E_08528

⌈The value of the `windowSizeValid`attribute shall be greater or equal to 1.⌋

[PRS_E2E_01458] Dependency between `windowSizeInvalid` and `windowSizeValid`

Replaces: PRS_E2E_CONSTR_06301
Upstream requirements: RS_E2E_08528

⌈The following restriction shall be respected: `windowSizeInvalid <= windowSizeValid`⌋

[PRS_E2E_01459] Dependency between `windowSizeInit` and `windowSizeValid`
Replaces: PRS_E2E_CONSTR_06302
Upstream requirements: RS_E2E_08528

⌈The following restriction shall be respected: `windowSizeInit <= windowSizeValid`⌋

[PRS_E2E_01460] Dependency between `maxErrorStateValid`
Replaces: PRS_E2E_CONSTR_03177
Upstream requirements: RS_E2E_08528

⌈`maxErrorStateInit` and `maxErrorStateInvalid` The following restriction shall be respected: `maxErrorStateValid >= maxErrorStateInit >= maxErrorStateInvalid >= 0`⌋

[PRS_E2E_01461] Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid`
Replaces: PRS_E2E_CONSTR_03178
Upstream requirements: RS_E2E_08528

⌈The following restriction shall be respected:
`1 <= minOkStateValid <= minOkStateInit <= minOkStateInvalid`⌋

[PRS_E2E_01462] Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeInit`
Replaces: PRS_E2E_CONSTR_03179
Upstream requirements: RS_E2E_08528

⌈The following restriction shall be respected:
`minOkStateInit + maxErrorStateInit <= windowSizeInit`⌋

[PRS_E2E_01463] Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSizeValid`
Replaces: PRS_E2E_CONSTR_03180
Upstream requirements: RS_E2E_08528

⌈The following restriction shall be respected: `minOkStateValid + maxErrorStateValid <= windowSizeValid`⌋

[PRS_E2E_01464] Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeInvalid`
Replaces: PRS_E2E_CONSTR_03181
Upstream requirements: RS_E2E_08528

⌈The following restriction shall be respected: `minOkStateInvalid + maxErrorStateInvalid <= windowSizeInvalid`⌋

9 Protocol usage and guidelines

This chapter contains requirements on usage of E2E Supervision when designing and implementing safety-related systems, which are depending on E2E communication protection and which are not directly related to some specific functionality. Note that Chapter 6 also provides several requirements on usages.

9.1 E2E and SOME/IP

For the combination of E2E communication protection with SOME/IP, there needs to be a common definition of the on-wire protocol. Depending on architecture properties, the implementing components need to be configured and used accordingly.

In general, all available E2E profiles can be used in combination with SOME/IP. However, they may have limitations, as for the maximum usable length of data, or being limited to fixed length messages.

The size of the E2E Header is dependent on the selected E2E profile.

[PRS_E2E_01444] Some/IP coverage

Replaces: PRS_E2E_UC_00239
Upstream requirements: RS_E2E_08540, RS_E2E_08541

⌈For profiles 1, 2, 4, 5, 6, 7, 11, and 22 the E2E CRC should be calculated over the following parts of the serialized SOME/IP message.

1. Request ID (Client ID / Session ID) [32 bit]
2. Protocol Version [8 bit]
3. Interface Version [8 bit]
4. Message Type [8 bit]
5. Return Code [8 bit]
6. Payload [variable size]

⌋

[PRS_E2E_01445] E2E for methods

Replaces: PRS_E2E_USE_00741
Upstream requirements: RS_E2E_08540

⌈For profiles 4m, 7m, 8m and 44m the E2E CRC shall be calculated over the following parts of the serialized SOME/IP message.

1. Payload [variable size]

⌋

[PRS_E2E_01446] Some/IP coverage

Replaces: PRS_E2E_UC_00238
Upstream requirements: RS_E2E_08540, RS_E2E_08541

⌈The E2E header should be placed after the Return Code depending on the chosen Offset value. The default Offset is 64 bit, which puts the E2E header exactly after the Return Code.⌋

9.2 Client-Server Communication

[PRS_E2E_01465] transfer counter from request to response

Replaces: PRS_E2E_UC_00606
Upstream requirements: RS_E2E_08541

⌈When a client sends a request to the server, the server should use the received counter as sequence counter for the response, no matter if regular response or error response.⌋

Some special profile for Client Server communication were created. These profiles are identified by letter "m" in their names. They contain special functionality for the use in Client-Server communication (see Chapter 6.4).

However, all other profiles not mentioned in Chapter 4.1.4 (see Chapter 6.3) can also be used for Client-Server communication. In this case the following has to be considered:

[PRS_E2E_01447] MaxDeltaCounter for Client-Server Communication (server)

Replaces: PRS_E2E_CONSTR_06300
Upstream requirements: RS_E2E_08528

⌈For Client-Server Communication the MaxDeltaCounter on server-side shall be set to the maximum of the value range of the sequence counter.⌋

[PRS_E2E_01448] MaxDeltaCounter for Client-Server Communication (client)

Replaces: PRS_E2E_CONSTR_06303
Upstream requirements: RS_E2E_08528

⌈For Client-Server Communication the MaxDeltaCounter on client-side shall be set to 1.⌋

Due to different send intervals of Clients the used counters will increment in a different rate, therefore counter jumps are inevitable. Due to the usage of the maximum value of the counter range all possible jumps are valid.

Nonetheless the server could receive the same counter from two or more clients, which will raise a E2E_P_REPEATED error on the server side. Since the E2E-protection of Client-Server Communication cannot detect counter related errors reliably on the

server side, the potentially raised E2E_P_REPEATED status code from the check function can be interpreted as E2E_P_OK.

To detect specific communication failure modes e. g. loss a deadline monitoring on client side is required. Since the request and the response counter have to be equal, no deadline monitoring is possible via the E2E Protocol, this has to be implemented by the E2E Protocol user.

9.3 Periodic use of E2E check

[PRS_E2E_01449] cyclic call FTTI

Replaces: PRS_E2E_UC_00325
Upstream requirements: RS_E2E_08528

⌈The E2E check function should be invoked at least once within FTTI (FTTI is for the safety goals from which the requirements for this E2E checks are derived).⌋

9.4 Error handling

The E2E Supervision itself does not handle detected communication errors. It only detects such errors for single received data elements and returns this information to the callers (e.g. SW-Cs), which have to react appropriately. A general standardization of the error handling of an application is usually not possible.

[PRS_E2E_01450] cyclic reception

Replaces: PRS_E2E_UC_00235
Upstream requirements: RS_E2E_08528

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

9.5 Maximal lengths of Data, communication buses

The length of the message and the achieved hamming distance for a given CRC are related. To ensure the required diagnostic coverage the maximum length of data elements protected by a CRC needs to be selected appropriately. The E2E profiles are intended to protect inter-ECU communication with lengths as listed in Table 9.1

All length values stated in this section are based on assumptions on suitable hamming distances for specific scenarios, without explicitly listing those assumptions. As such, actual suitable values may differ based on the use case scenarios.

E2E Profile	Suggested maximum applicable length including control fields for inter-ECU communication

E2E Profile 1 and 11	32B
E2E Profile 2 and 22	32B
E2E Profile 4 and 4m	4 kB
E2E Profile 5	4 kB
E2E Profile 6	4 kB
E2E Profile 7 and 7m	4 MB

Table 9.1: E2E maximum data length

In E2E Profiles 1 and 2, the Hamming Distance is 2, up to the given lengths. Due to 8 bit CRC, the burst error detection is up to 8 bits.

9.6 Functional Safety Requirements

When using E2E Supervision, the designer of the functional or technical safety concept of a particular system using E2E Supervision should evaluate the maximum permitted length of the protected Data in that system, to ensure an appropriate error detection capability.

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

If the protected data length exceeds the network bus frame limit (or payload limit), the data can be segmented on the sender side after the E2E communication protection, and be assembled on the receiver side before the E2E evaluation. The possible faults happening during segmentation/desegmentation can be considered as "corruption of information".

In other words, SW-C shall be able to tolerate the reception of one erroneous data element, whose error was not detected by the E2E Supervision. What is not required is that an SW-C tolerates two consecutive undetected erroneous data elements, because it is enough unlikely that two consecutive Data are wrong AND that for both Data the error remains undetected by the E2E Supervision.

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

E2E Profile 1 with E2E_P01DataIDMode = E2E_P01_DATAID_BOTH and E2E Profile 11 with E2E_P11DataIDMode = E2E_P11_DATAID_BOTH 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 SWS_E2ELibrary items PRS_E2E_UC_00072 and PRS_E2E_UC_00073 [4].

9.7 Message Layout

This chapter provides some requirements and recommendations on how safety-related messages shall or should be defined. These recommendations can be also extended to non-safety-related data transmissions.

9.7.1 Alignment of signals to byte limits

This chapter provides some requirements and recommendation on how safety-related data structures shall or can be defined. They could also be extended to non-safety-related data structures if found adequate.

[PRS_E2E_01451] Alignment

Replaces: PRS_E2E_UC_00062
Upstream requirements: RS_E2E_08528

⌈When using E2E Profiles, messages that have length < 8 bits should be allocated to one byte of a message, i.e. they should not span over two bytes.⌋

[PRS_E2E_01452] Alignment

Replaces: PRS_E2E_UC_00063
Upstream requirements: RS_E2E_08528

⌈When using E2E Profiles, messages that have length >= 8 bits should start or finish at the byte limit of a message⌋

[PRS_E2E_01453] length of data

Replaces: PRS_E2E_UC_00320
Upstream requirements: RS_E2E_08528

⌈When using E2E Profiles, the length of the data to be protected should be multiple of 8 bits.⌋

The previous recommendations cause that signals of type uint8, uint16 and uint32 fit exactly to respectively one, two or four byte(s) of a message. These recommendations also cause that for uint8, uint16 and uint32, the bit offsets are a multiple of 8.

Figure Figure 9.1 is an example of signals (CRC, Alive and Sig1) that are not aligned to the Byte limits.

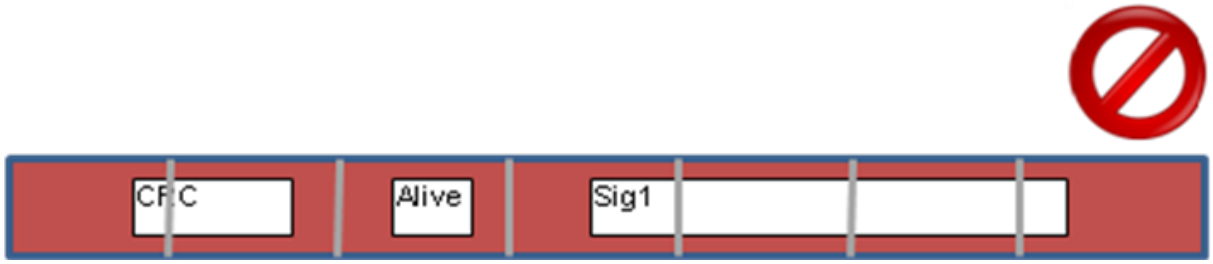

Figure 9.1: Example for alignment not following recommendations

9.7.2 Unused bits

It can happen that some bits in a protected data structure are unused. In such a case, the sender does not send signals represented by these bits, and the receiver does not expect to receive signals represented by these bits. In order to have a systematically defined data structure and sender-receiver behavior, the unused bits are set to the defined default value before calculation of the CRC.

[PRS_E2E_01454] unused areas of messages

Replaces: PRS_E2E_UC_00173
Upstream requirements: RS_E2E_08528

⌈Any sender, which uses the E2E-Profiles, should fill all unused areas of a messages to a default value configured for the message).⌋

For signal based communication of I-PDU's this is defined in the system template parameter ISignalIpdu.unusedBitPattern. The attribute unusedBitPattern is actually an 8-bit Byte pattern. It can take any value from 0x00 to 0xFF. Often 0xFF is used.

9.7.3 Byte order (Endianness)

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

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

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

1. Sender converts the byte Endianness of the data

2. Sender copies the converted data on I-PDU (serializes the signal), while copying only used bits from the signals,
3. Sender COM delivers unaltered I-PDU to receiver COM (an I-PDU is just a byte array unaltered by lower layers of the network stack),
4. Receiver COM converts the Endianness of the signals in the received I-PDU (if configured). It may also do the sign extension (if configured),
5. Receiver COM returns the converted signals.

To achieve high level of interoperability, the automotive networks recommend a particular byte order, which is depicted in table Table 9.2.

Network	Byte order
FlexRay	Little Endian
CAN	Little Endian
LIN	Little Endian
TCP/IP	Big Endian
Byteflight	Big Endian
MOST	Big Endian

Table 9.2: Networks and their byte order

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

9.8 Configuration constraints on Data IDs

9.8.1 Data IDs

To be able to verify the identity of the data elements or signal groups, none of two are allowed to have the same Data ID (E2E Profiles 1, 4, 5, 6, 7, 11, 4m, 7m) or same DataIDList[] (E2E Profile 2, 22) within one system of communicating ECUs.

It is recommended that the value of the Data ID be assigned by a central authority rather than by the developer of the software-component. The Data IDs are defined in Software Component Template, and then realized in E2E_PXXConfig structures.

Note: For Profile 1 requirement (PRS_E2E_UC_00071) may not be sufficient in some cases, because Data ID is longer than CRC, which results with additional requirements PRS_E2E_UC_00072 and PRS_E2E_UC_00073. In Case of Profile 1 the ID can be encoded in CRC by double Data ID configuration (both bytes of Data ID are included in CRC every time), or in alternating Data ID configuration (high byte or low byte of Data ID are put in CRC alternatively, depending of parity of Counter), there are different additional requirements/constraints described in the sections below.

9.8.2 Double Data ID configuration of E2E Profile 1 and 11

In E2E Profiles 1 and 11, the CRC is 8 bits, whereas Data ID is 16bits. In the double Data ID configuration (both bytes of Data ID are included in CRC every time), like it is in the E2E variant 1A, all 16 bits are always included in the CRC calculation. In consequence, two different 16 bit Data IDs DI1 and DI2 of data elements DE1 and DE2 may have the same 8 bit CRC value. Now, a possible failure mode is for example that a gateway incorrectly routes a safety-related signal DE1 to the receiver of DE2. The receiver of DE2 receives DE1, but because the DI1 and DI2 are identical, the receiver might accept the message (this assumes that by accident the counter was also correct and that possibly data length was the same for DE1 and DE2).

To resolve this, there are additional requirements limiting the usage of ID space. Data elements with ASIL B and above shall have unique CRC over their Data ID, and signals having ASIL A requirements shall have a unique CRC over their Data IDs for a given data element/signal length.

The above requirement limits the usage of Data IDs of data having ASIL B, C, D to 255 distinct values in a given ECU, but gives the flexibility to define the Data IDs within the 16-bit naming space.

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

9.8.3 Alternating Data ID configuration of E2E Profile 1 and 11

In the alternating Data ID configuration, either high byte or low byte of Data ID is put in CRC alternatively, depending of parity of Counter. In this configuration, two consecutive Data are needed to verify the data identity. This is not about the reliability of the checksum or software, but really the algorithm constraint, as on every single Data only a single byte of the Data ID is transmitted and therefore it requires two consecutive receptions to verify the Data ID of received Data.

9.8.4 Nibble configuration of E2E Profile 1 and 11

In the nibble Data ID configuration of E2E Profile 1 and 11, the low byte is not transmitted, but included in the CRC. Because the low byte has the length of 8 bits, it is the same as the CRC.Therefore, if two Data IDs are different in the low byte, this results with a different CRC over the Data ID low byte.

[PRS_E2E_01455] high nibble value at profiles 1 and 11

Replaces: PRS_E2E_UC_00308
Upstream requirements: RS_E2E_08528

⌈Any user of Profile 1 or 11 in Nibble Data ID configuration should ensure that:

1. the high nibble of high byte of Data ID is equal to 0

2. the low nibble of high byte of Data ID is within the range 0x1..0xE (to avoid collisions with other E2E Profile 1 configurations that have 0x0 on this nibble, and to exclude the invalid value 0xF).
3. The low byte of Data ID is different to low byte of any Data ID present in the same bus that uses E2E Profile in Double Data ID configuration.

⌋

[PRS_E2E_01456] Data ID values

Replaces: PRS_E2E_UC_00317
Upstream requirements: RS_E2E_08527

⌈When using E2E Profile Variants 1/11A and 1/11C in one bus/system, the following should be respected:

1. 1/11A data should use IDs that are < 256 (this means high byte shall be always = 0)
2. 1/11C data should use IDs that are >= 256 (this means high byte is always != 0) and < 4‘096 (0x10’00 - it means they fit to 12 bits).
3. Any low byte of 1/11C data id should be different to any low byte of 1/11A data ID.

⌋

Thanks to the Data ID distribution according to the above requirement, addressing errors can be detected: in particular, it can be detected when 1/11C message arrives to 1/11A destination. If 1/11C message receives to a 1/11A destination, then the CRC check will pass if low byte of the sent 1/11C message equals to the expected 1/11A address - and this is excluded by the above requirement.

Example: 1A may use addresses 0 to 199, while 1C may use addresses where low byte is 200 to 255 and high byte is between 1 and 15. This allows to use additional (256-200)*15 = 840 Data IDs. Once it is known, the corresponding E2E Library CRC routines should be used.

A Usage and generation of DataID Lists for E2E profile 2 and 22

An appropriate selection of DataIDs for the DataIDList in E2E Profiles 2 and 22 allows increasing the number of messages for which detection of masquerading is possible. The DataID is used when calculating the CRC checksum of a message, whereas the DataID is not part of the transmitted message itself, i.e. the message received by the receiver does not contain this information.

Any receiver of the intended message needs to know the DataID a priori. The performed check of the received CRC at the receiver side does only match if and only if the assumed DataID on the receiver side is identical to the DataID used at the sender side. Thus, the DataID allows protecting messages against masquerading. It is important that the used DataID is known solely by the intended sender and the intended receiver.

With a constant DataID (independent of the Counter) the maximum number of messages that can be protected independently using E2E Profile 2 is limited by the length of the CRC (i.e. with a CRC length of 8 bits the number of independent DataID is 2 * 8 = 256, this equates to the maximum number of independent messages for detection of masquerading).

However, E2E Profiles 2 and 22 uses a method to allow more messages to be protected against masquerading by exploiting the prerequisite that a single erroneously received message content does not violate the safety goal (a basic assumption taken in the design of applications of receiving SW-Cs).

The basic idea in E2E Profiles 2 and 22 is to use a DataIDList with several DataIDs that are selected in a dynamic behavior for the calculation of the CRC checksum. The DataID is determined by selecting one element out of DataIDList, using the value of Counter as an index (for detailed description see E2E profile 2).

The examples given below were selected to show two exemplary use cases. It is demonstrated how the detection of masquerading is performed.

Although the examples take some assumptions on the configuration, the argumentation is valid without loss of generality. For sake of simplicity, these additional constraints are not explained in the following examples.

A.1 Example A (persistent routing error)

A.1.1 Assumptions

Consider a network with one or more nodes as sender (messages A to F) and one node as the intended receiver of the safety relevant message (message B). The messages are configured to use the DataIDList as shown in Figure A.1 and Figure A.2.

Sender-ECU		DataIDList															
		DataID for Counter =															
	message	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sender	A	177	103	29	206	132	58	235	161	87	13	190	116	42	219	145	71
Sender	B	146	41	187	82	228	123	**18**	164	59	205	100	246	141	36	182	77
Sender	C	102	204	55	157	8	110	212	63	165	16	118	220	71	173	24	126
Sender	D	225	199	173	147	121	95	69	43	17	242	216	190	164	138	112	86
Sender	E	181	112	43	225	156	87	**18**	200	131	62	244	175	106	37	219	150
Sender	F	244	244	244	244	244	244	244	244	244	244	244	244	244	244	244	244 ←special case of static DataID

Figure A.1: Example for alignment not following recommendations -> Sender-ECU IDs

Receiver-ECU		DataIDList															
		Counter =															
	message	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Receiver	B	146	41	187	82	228	123	**18**	164	59	205	100	246	141	36	182	77

Figure A.2: Example for alignment not following recommendations -> Receiver-ECU IDs

In the example of Figure A.3 it is assumed that a routing error occurs at a specific point in time. All messages are of same length. The routing error persists until it is detected. For instance a bit flip of the routing table in a gateway could lead to such a constant misrouting. It is further assumed that the senders of messages B and E have the same sequence counter (worst case situation for detection in the receiver).

The receiver should only receive message B and expects therefore the DataIDs of DataIDList of message B. Every time the expected DataID matches with the used DataID in the CRC-protected message, the result of the CRC check will be valid. In any other case the CRC checksum in the message differs from the expected CRC result and the outcome of the CRC check is not valid.

A.1.2 Solution

As depicted, the first routing error occurs when both senders reach Counter = 6. Since the DataIDList in both senders have DataID = 18 for Counter = 6, the receiver will not detect the erroneously routed message of sender E. However, for any other Counter the values of DataIDs do not match, thus the CRC check in the receiver will be not valid. With this, it is obvious that the misrouting is detected at least for the second received misrouted message (even if some messages were not received at all).

	Sender of B		Sender of E		Receiver expects message B				
	Counter	DataID	Counter	DataID	Counter	DataID used	check	DataID expected	result of CRC-Check
	0	146	0	181	0	146	=	146	valid
	1	41	1	112	1	41	=	41	valid
	2	187	2	43	2	187	=	187	valid
	3	82	3	225	3	82	=	82	valid
	4	228	4	156	4	228	=	228	valid
	5	123	5	87	5	123	=	123	valid
here 1st ⟶	6	18	**6**	**18**	**6**	**18**	=	18	erroneously undetected! (valid)
routing error	7	164	**7**	**200**	**7**	**200**	≠	164	error detected (not valid)
	8	59	**8**	**131**	**8**	**131**	≠	59	error detected (not valid)
	9	205	**9**	**62**	**9**	**62**	≠	205	error detected (not valid)
	10	100	**10**	**244**	**10**	**244**	≠	100	error detected (not valid)
	11	246	**11**	**175**	**11**	**175**	≠	246	error detected (not valid)
	12	141	**12**	**106**	**12**	**106**	≠	141	error detected (not valid)
	13	36	**13**	**37**	**13**	**37**	≠	36	error detected (not valid)
	14	182	**14**	**219**	**14**	**219**	≠	182	error detected (not valid)
	15	77	**15**	**150**	**15**	**150**	≠	77	error detected (not valid)
	…	…	…	…	…	…	…	…	…
	5	123	**5**	**87**	**5**	**87**	≠	123	error detected (not valid)

Figure A.3: Example A configuration

A.1.3 Example B (forbidden configuration)

Not every DataIDList is allowed to be used for every message length. A short explanation to demonstrate this is shown in this example.

Consider a message G with a total length of 8 bytes. Both, sender and receiver are configured to use the DataIDList depicted in Figure A.4

Receiver-ECU		**DataIDList**															
		Counter =															
	message	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Receiver	G	73	144	215	35	106	177	248	68	139	210	30	101	172	243	63	134

Figure A.4: Example B configuration (forbidden configuration)

Without loss of generality the payload is assumed to be [22,33,44,55,66,77].

For the defined CRC generator polynomial in profile 2 and 22 the CRC checksums are as follows:

```
Counter Data                    DataID CRC-result
CRC( 0,22,33,44,55,66,77, 73) = 114
CRC( 1,22,33,44,55,66,77,144) = 197
CRC( 2,22,33,44,55,66,77,215) =  66
CRC( 3,22,33,44,55,66,77, 35) =  66
CRC( 4,22,33,44,55,66,77,106) = 207
CRC( 5,22,33,44,55,66,77,177) =  38
CRC( 6,22,33,44,55,66,77,248) =  20
CRC( 7,22,33,44,55,66,77, 68) = 165
CRC( 8,22,33,44,55,66,77,139) = 120
CRC( 9,22,33,44,55,66,77,210) =  44
CRC(10,22,33,44,55,66,77, 30) = 110
CRC(11,22,33,44,55,66,77,101) =  23
```

```
CRC(12,22,33,44,55,66,77,172) = 121
CRC(13,22,33,44,55,66,77,243) = 207
CRC(14,22,33,44,55,66,77, 63) = 141
CRC(15,22,33,44,55,66,77,134) = 175
```

One can see that DataID = 215 for Counter = 2 leads to the same CRC checksum as DataID = 35 for Counter = 3. Moreover, DataID = 106 for Counter = 4 leads to the same CRC checksum as DataID = 243 for Counter = 13.

A routing error of a non-CRC-protected message with constant payload and a sequence counter could be undetected at the receiver side if

1. the first routing error occurs at Counter = 2 and is persistent, or
2. the routing error occurs only at Counter = 4 and Counter = 13.

In both cases the second masquerading error is not detected.

Thus, the considered DataIDList of message G in Figure A.4 shall not be used for messages with a total length of 8 bytes. (Remember: the DataID itself is never transmitted on the bus).

A.2 Conclusion

The proposed method with dynamic DataIDs for CRC calculation allows protecting significantly (several orders of magnitude) more messages against masquerading than with a static DataID.

The set of DataIDList needs to be generated with appropriate care to utilize the strength of the shown method. Every DataIDList is only allowed to be assigned once to a message within the network/system. The message length needs to be considered in the assignment process since not every DataIDList is allowed to be used for every message length.

A.3 DataID List example

This section presents a part of exemplary DataIDList. The example has 500 lines, which means that this enables to identify 500 different data.

This DataIDList of 9 subtables has been selected and tested with appropriate care to comply with current safety standards. Every user of the provided DataIDLists is responsible to check if the following list is suitable to fulfill his constraints of the intended target network.

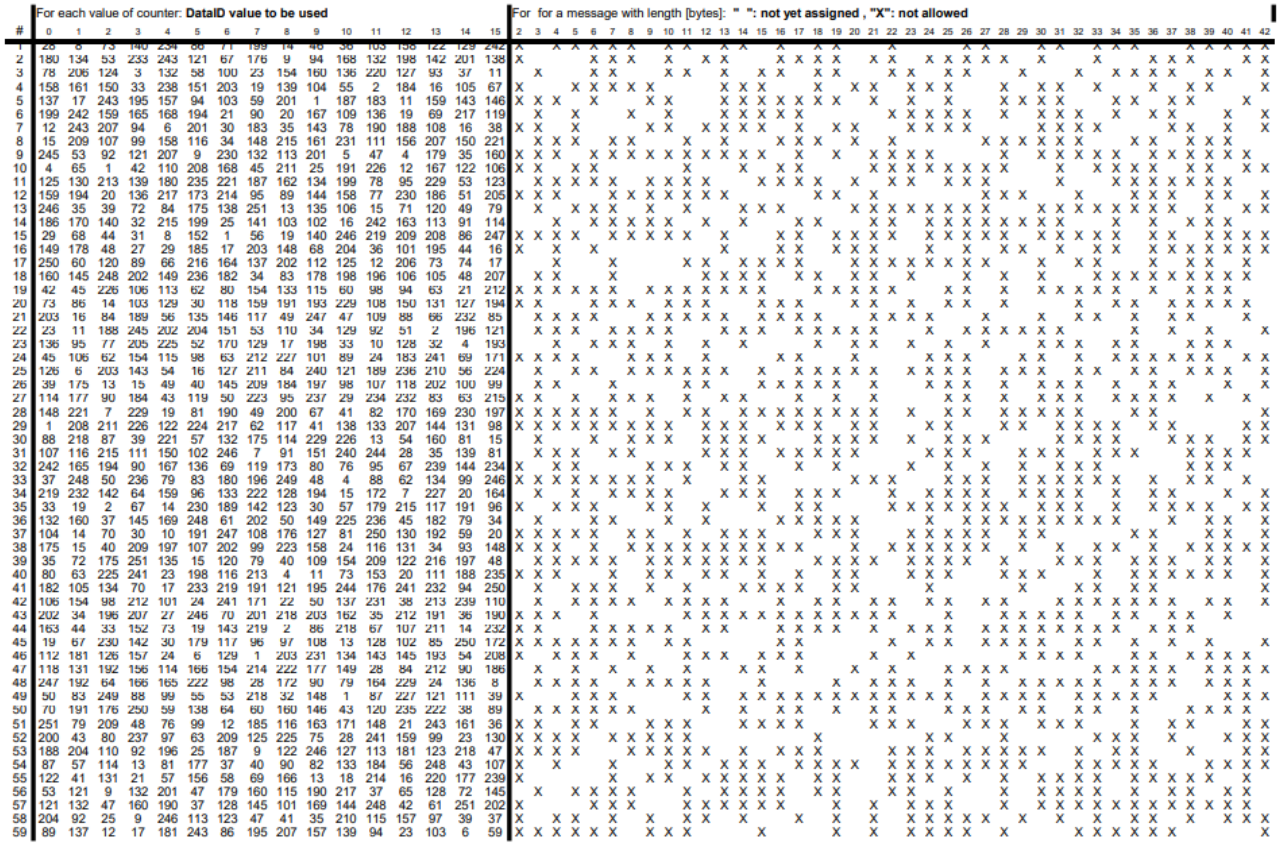
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed
#

Figure A.5

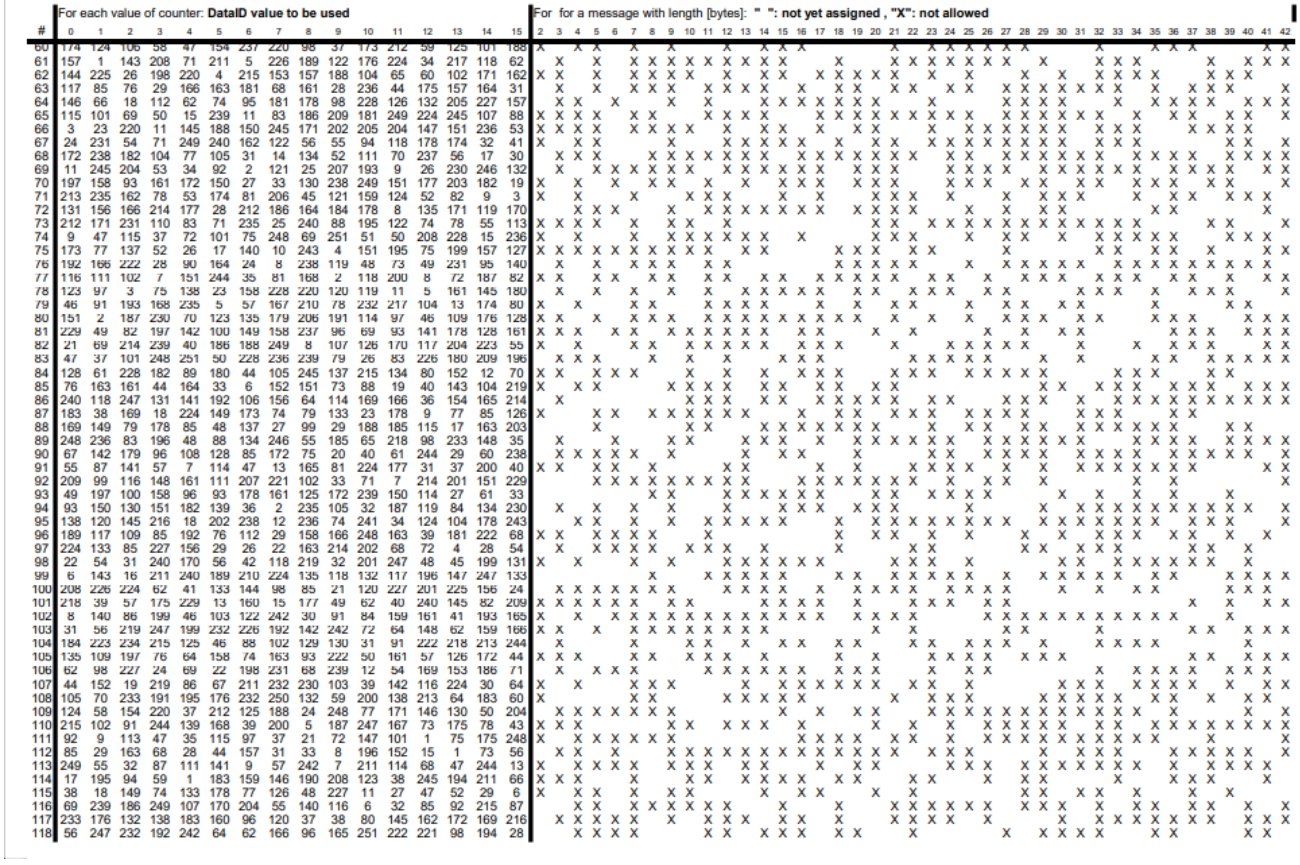
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed
#

Figure A.6

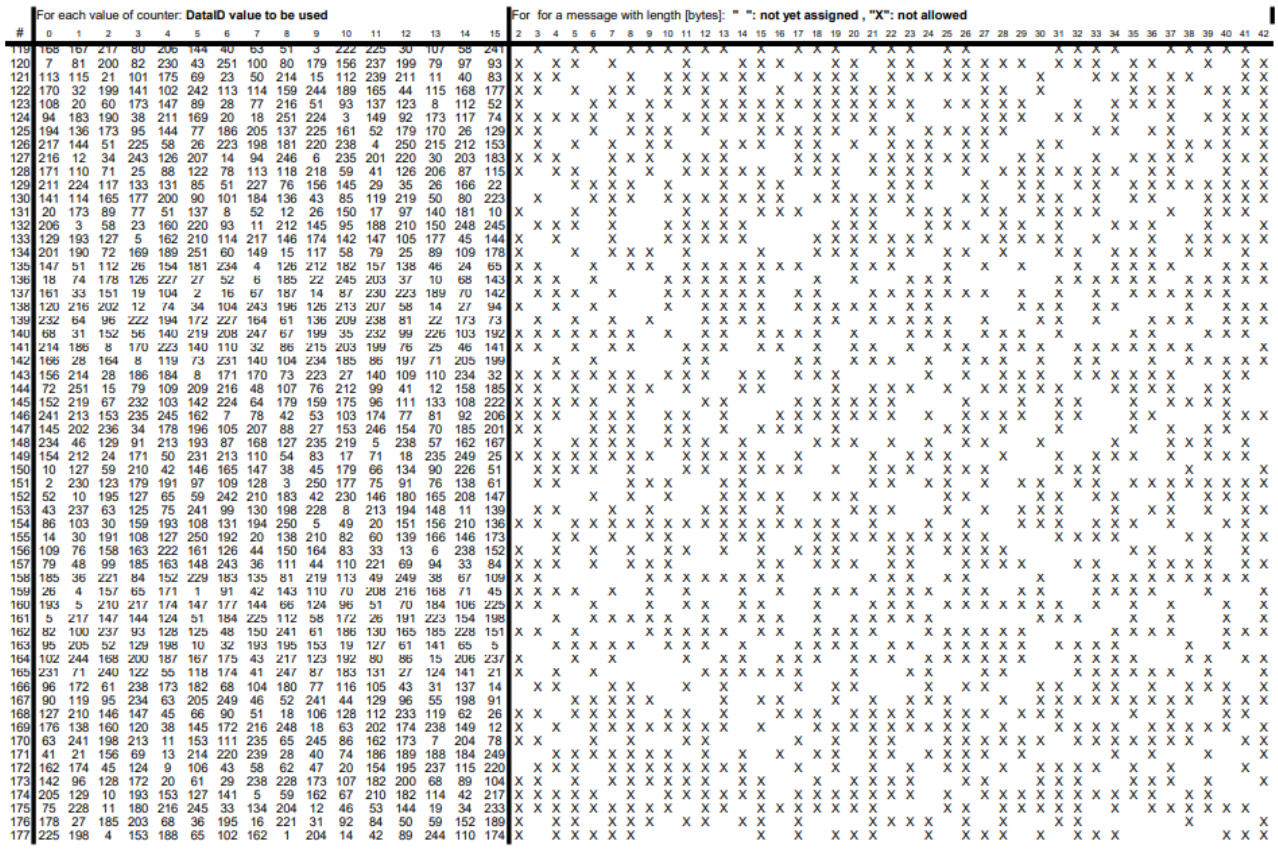
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed
#

Figure A.7

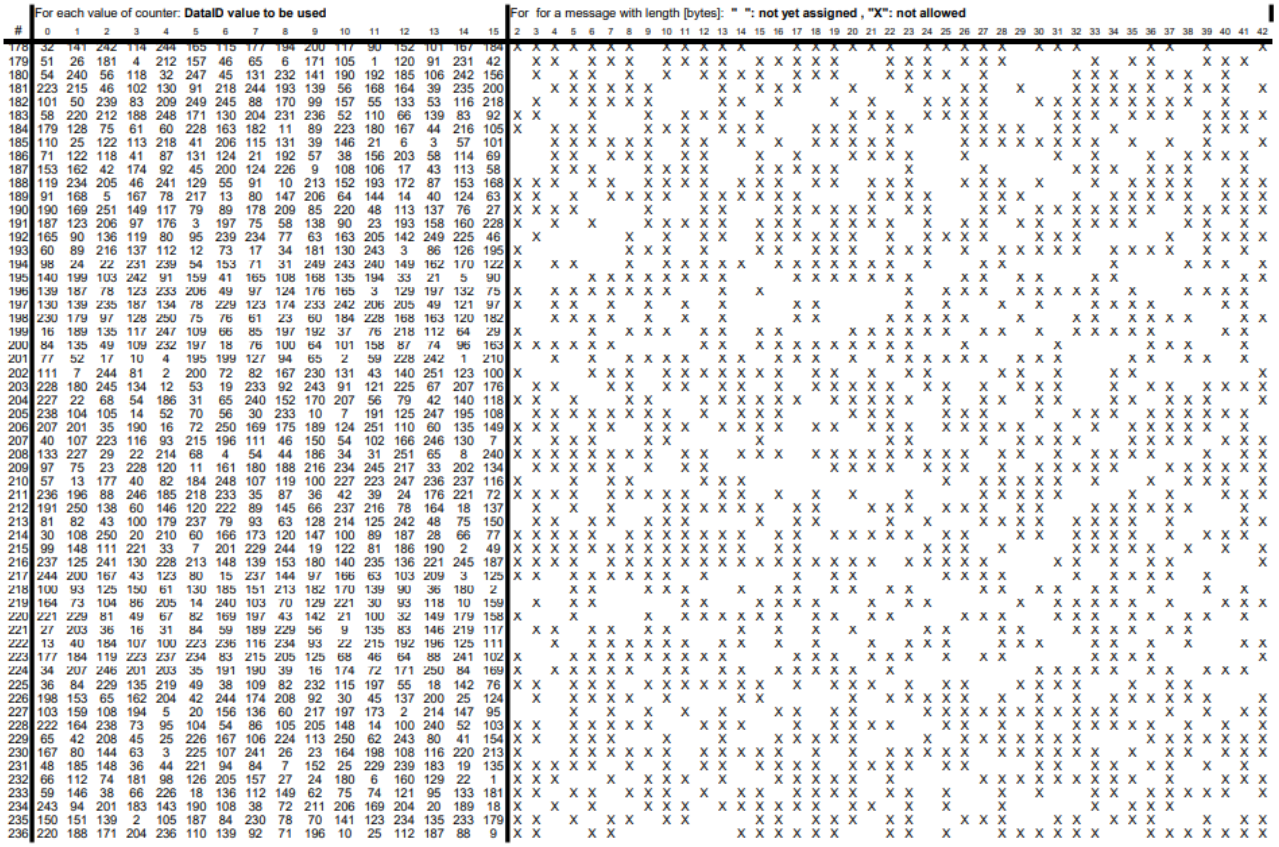
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed
#
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42

Figure A.8

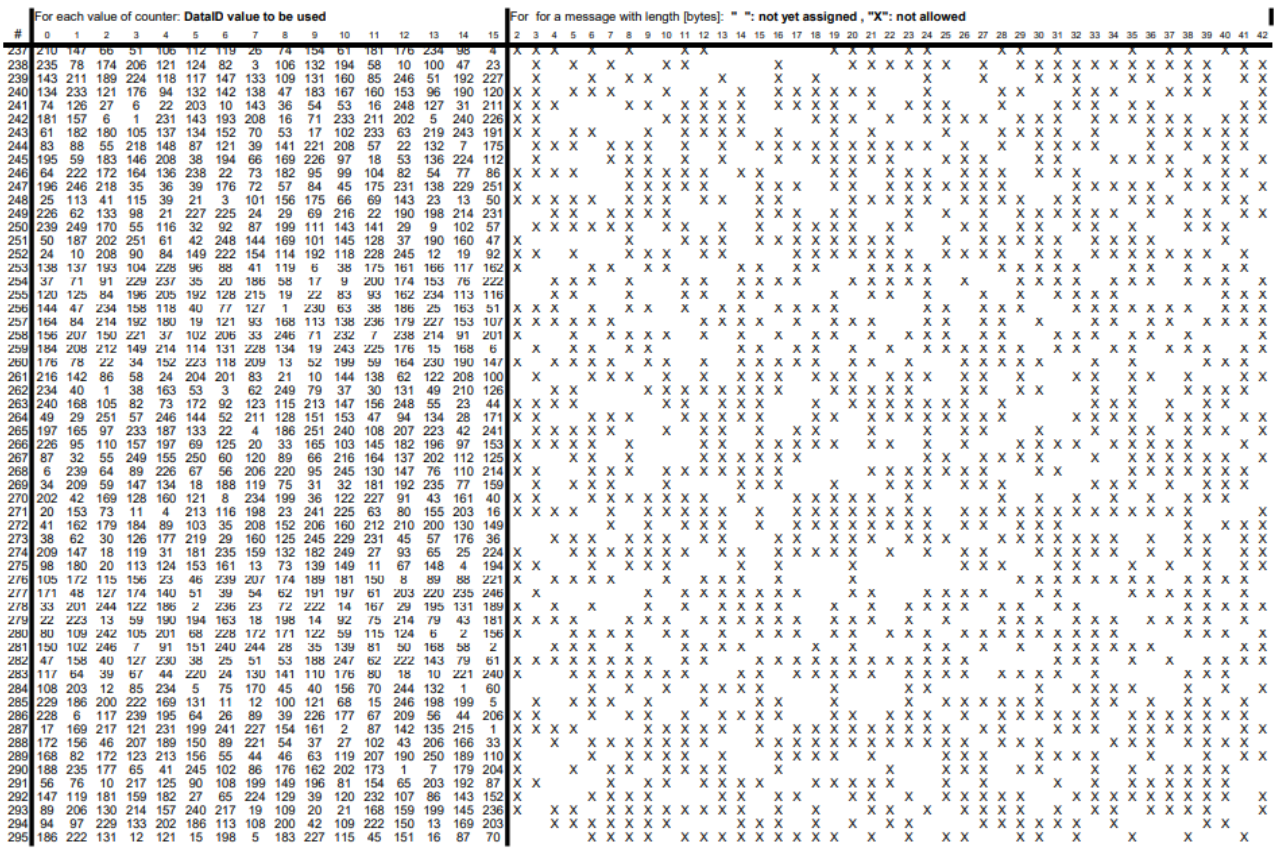
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed
#

Figure A.9

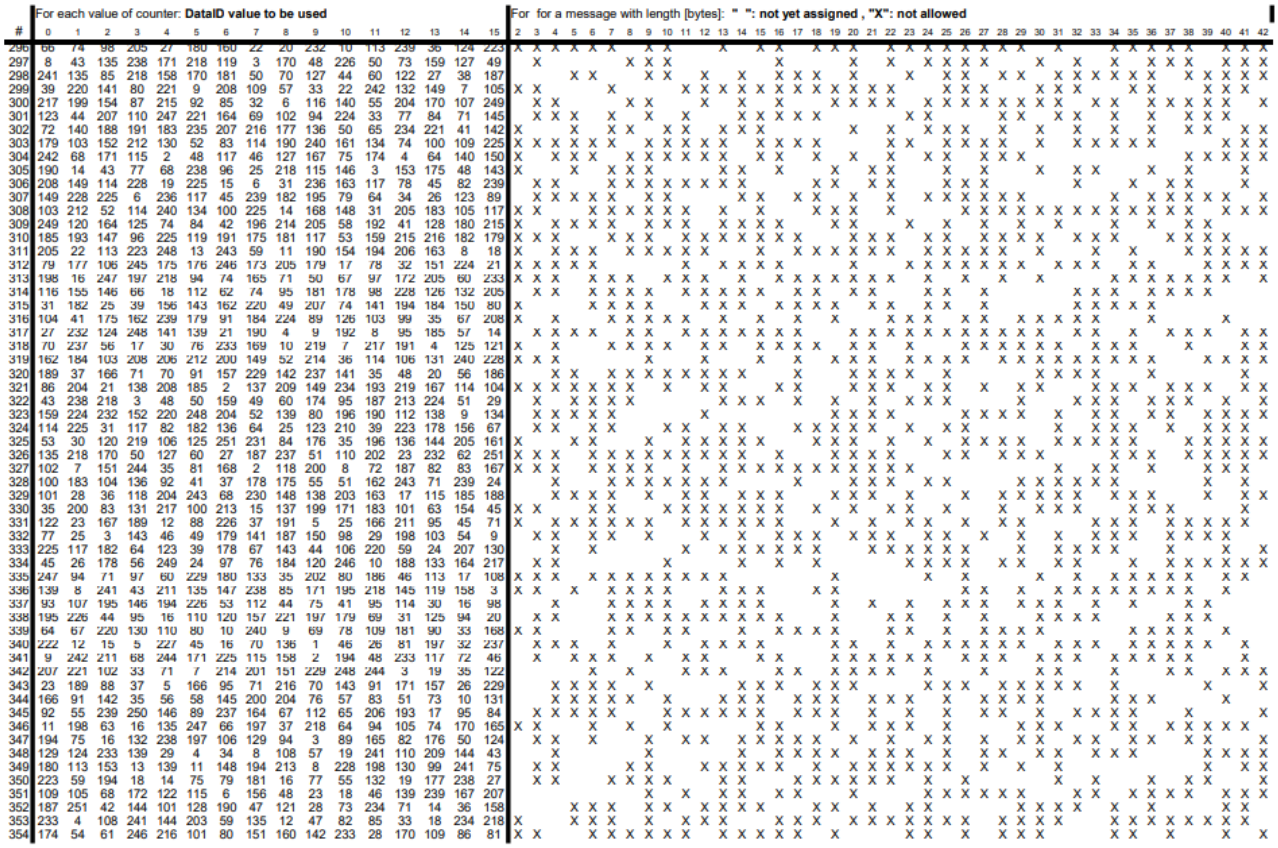
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed

Figure A.10

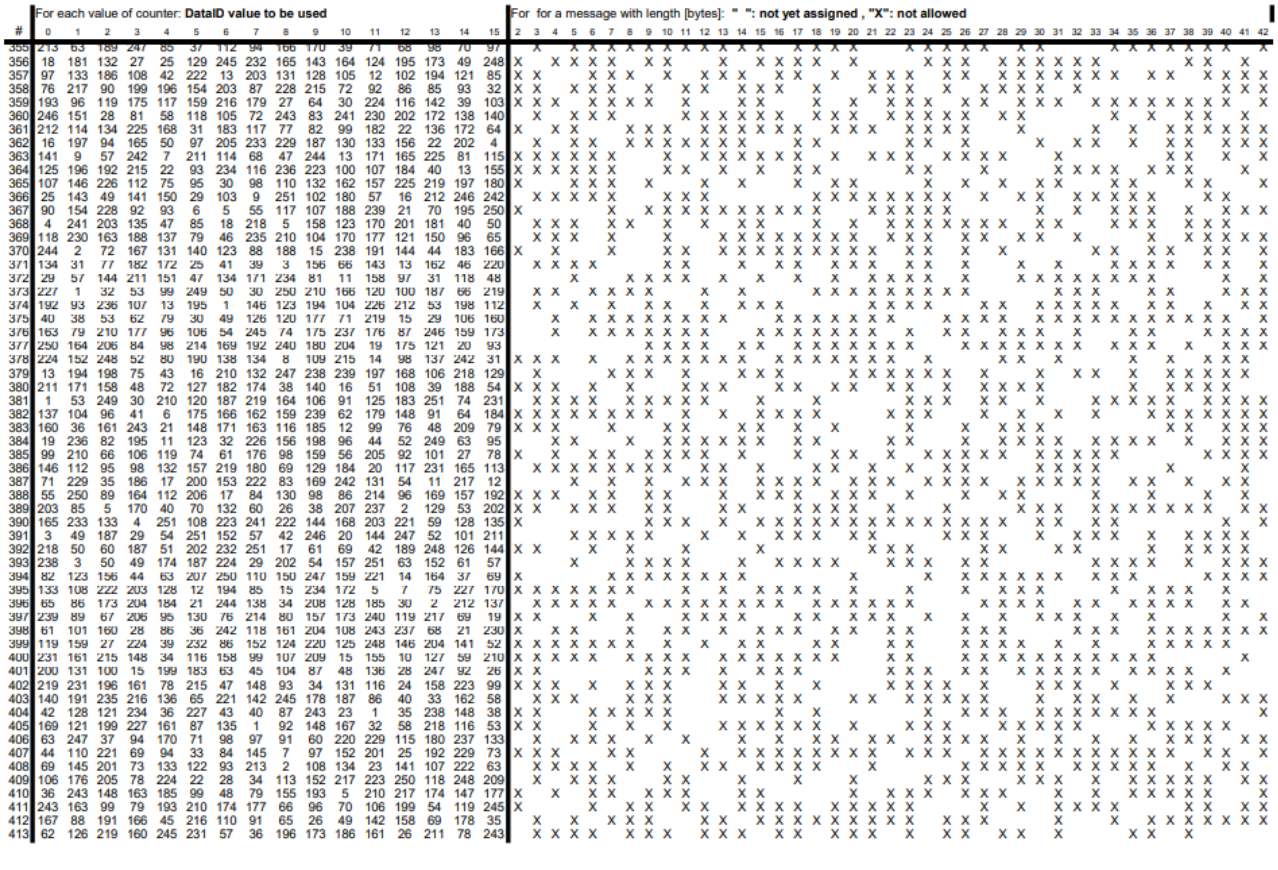
For each value of counter: DataID value to be used
For for a message with length [bytes]: " ": not yet assigned , "X": not allowed
#

Figure A.11

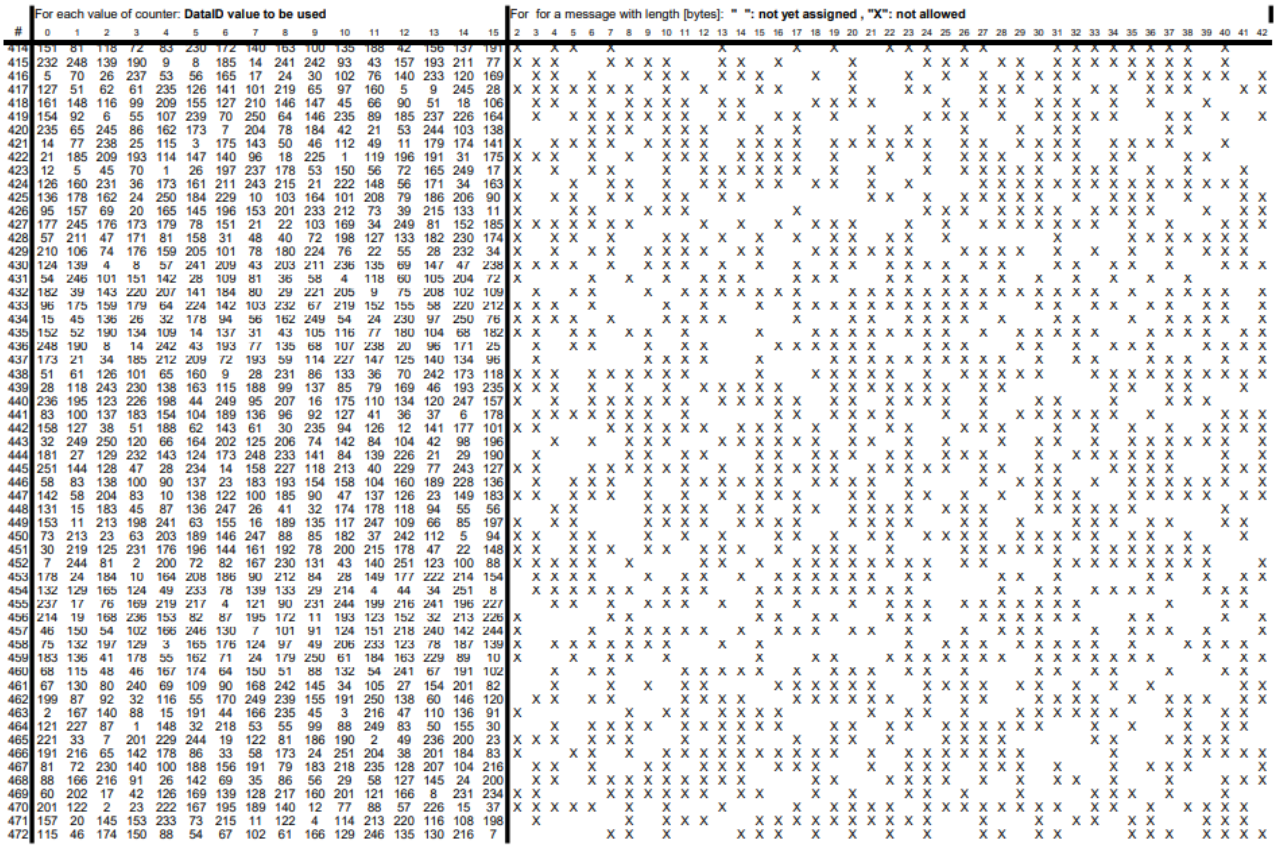

Figure A.12

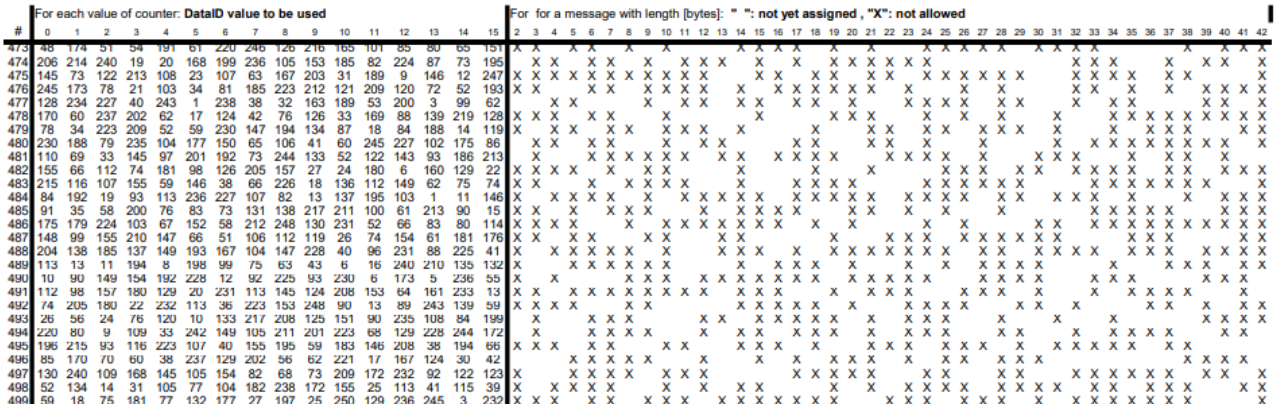

Figure A.13

B Change history of AUTOSAR traceable items

B.1 Traceable item history of this document according to AUTOSAR Release R19-11

B.1.1 Added Constraints

Number	Heading
Constraint [PRS_E2E_CONSTR_03176] in Chapter 8.1.1	Value range of `windowSizeValid`
Constraint [PRS_E2E_CONSTR_03177] in Chapter 8.1.1	Dependency between `maxErrorStateValid`, `maxErrorStateInit` and `maxErrorStateInvalid`
Constraint [PRS_E2E_CONSTR_03178] in Chapter 8.1.1	Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid`
Constraint [PRS_E2E_CONSTR_03179] in Chapter 8.1.1	Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeInit`
Constraint [PRS_E2E_CONSTR_03180] in Chapter 8.1.1	Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSizeValid`
Constraint [PRS_E2E_CONSTR_03181] in Chapter 8.1.1	Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeInvalid`
Constraint [PRS_E2E_CONSTR_06300] in Chapter 9.2	MaxDeltaCounter for Client-Server Communication
Constraint [PRS_E2E_CONSTR_06301] in Chapter 8.1.1	Dependency between `windowSizeInvalid` and `windowSizeValid`
Constraint [PRS_E2E_CONSTR_06302] in Chapter 8.1.1	Dependency between `windowSizeInit` and `windowSizeValid`

Table B.1: added Constraints in R19-11

B.1.2 Changed Constraints

N/A

B.1.3 Deleted Constraints

N/A

B.1.4 Added Specification Items

N/A

B.1.5 Changed Specification Items

N/A

B.1.6 Deleted Specification Items

N/A

B.2 Traceable item history of this document according to AUTOSAR Release R20-11

B.2.1 Added Constraints in R20-11

none

B.2.2 Changed Constraints in R20-11

none

B.2.3 Deleted Constraints in R20-11

none

B.2.4 Added Specification Items in R20-11

Number	Heading
[PRS_E2E_00640]	
[PRS_E2E_00641]	Mapping Profile 1 to State Machine
[PRS_E2E_00644]	
[PRS_E2E_00645]	
[PRS_E2E_00646]	
[PRS_E2E_00647]	
[PRS_E2E_00648]	

▽

△

Number	Heading
[PRS_E2E_00649]	
[PRS_E2E_00650]	
[PRS_E2E_00651]	
[PRS_E2E_00652]	
[PRS_E2E_00653]	
[PRS_E2E_00654]	
[PRS_E2E_00655]	
[PRS_E2E_00656]	
[PRS_E2E_00657]	
[PRS_E2E_00658]	
[PRS_E2E_00659]	
[PRS_E2E_00660]	
[PRS_E2E_00661]	
[PRS_E2E_00662]	
[PRS_E2E_00663]	
[PRS_E2E_00664]	
[PRS_E2E_00665]	
[PRS_E2E_00666]	
[PRS_E2E_00667]	
[PRS_E2E_00668]	
[PRS_E2E_00669]	
[PRS_E2E_00670]	Mapping Profile 2 to State Machine
[PRS_E2E_00673]	
[PRS_E2E_00675]	
[PRS_E2E_00676]	
[PRS_E2E_00677]	
[PRS_E2E_00678]	
[PRS_E2E_00679]	
[PRS_E2E_00680]	
[PRS_E2E_00681]	
[PRS_E2E_00682]	
[PRS_E2E_00683]	
[PRS_E2E_00684]	
[PRS_E2E_00685]	
[PRS_E2E_00686]	
[PRS_E2E_00687]	
[PRS_E2E_00688]	
[PRS_E2E_00689]	

▽

△

Number	Heading
[PRS_E2E_00690]	
[PRS_E2E_00691]	
[PRS_E2E_00692]	
[PRS_E2E_00693]	
[PRS_E2E_00694]	
[PRS_E2E_00695]	
[PRS_E2E_00696]	
[PRS_E2E_00697]	
[PRS_E2E_00698]	
[PRS_E2E_00699]	
[PRS_E2E_00700]	
[PRS_E2E_00701]	
[PRS_E2E_00702]	
[PRS_E2E_00703]	
[PRS_E2E_00704]	
[PRS_E2E_00705]	
[PRS_E2E_00706]	
[PRS_E2E_00707]	
[PRS_E2E_00708]	
[PRS_E2E_00709]	
[PRS_E2E_00710]	
[PRS_E2E_00711]	
[PRS_E2E_00712]	
[PRS_E2E_00713]	
[PRS_E2E_00714]	
[PRS_E2E_00715]	
[PRS_E2E_00716]	
[PRS_E2E_00717]	
[PRS_E2E_00718]	
[PRS_E2E_00719]	
[PRS_E2E_00720]	
[PRS_E2E_00721]	
[PRS_E2E_00722]	
[PRS_E2E_00723]	
[PRS_E2E_00724]	
[PRS_E2E_00725]	
[PRS_E2E_00726]	
[PRS_E2E_00727]	

▽

△

Number	Heading
[PRS_E2E_00728]	
[PRS_E2E_00729]	
[PRS_E2E_00730]	
[PRS_E2E_00731]	
[PRS_E2E_00732]	
[PRS_E2E_00733]	
[PRS_E2E_00734]	
[PRS_E2E_00735]	
[PRS_E2E_00736]	
[PRS_E2E_00737]	
[PRS_E2E_00738]	Mapping Profile 44 to State Machine
[PRS_E2E_00739]	
[PRS_E2E_00740]	
[PRS_E2E_00741]	
[PRS_E2E_00742]	
[PRS_E2E_00743]	
[PRS_E2E_00744]	
[PRS_E2E_00745]	
[PRS_E2E_00746]	
[PRS_E2E_00747]	
[PRS_E2E_00748]	
[PRS_E2E_00749]	
[PRS_E2E_00750]	
[PRS_E2E_00751]	
[PRS_E2E_00752]	
[PRS_E2E_00753]	
[PRS_E2E_00756]	
[PRS_E2E_00757]	
[PRS_E2E_00758]	
[PRS_E2E_00759]	
[PRS_E2E_00760]	
[PRS_E2E_00761]	
[PRS_E2E_00762]	
[PRS_E2E_00763]	
[PRS_E2E_00764]	
[PRS_E2E_00765]	
[PRS_E2E_00766]	
[PRS_E2E_00767]	

▽

△

Number	Heading
[PRS_E2E_00768]	Draft
[PRS_E2E_00769]	Draft
[PRS_E2E_00770]	Draft
[PRS_E2E_00771]	Draft
[PRS_E2E_00772]	
[PRS_E2E_00773]	
[PRS_E2E_00774]	
[PRS_E2E_00775]	
[PRS_E2E_00776]	
[PRS_E2E_00777]	
[PRS_E2E_00778]	
[PRS_E2E_00779]	
[PRS_E2E_00780]	
[PRS_E2E_00781]	
[PRS_E2E_00783]	
[PRS_E2E_00784]	
[PRS_E2E_00785]	
[PRS_E2E_00787]	
[PRS_E2E_00788]	
[PRS_E2E_00789]	
[PRS_E2E_00790]	
[PRS_E2E_00791]	
[PRS_E2E_00792]	
[PRS_E2E_00793]	
[PRS_E2E_00794]	
[PRS_E2E_00795]	
[PRS_E2E_00796]	
[PRS_E2E_00799]	
[PRS_E2E_00800]	
[PRS_E2E_00801]	
[PRS_E2E_00802]	
[PRS_E2E_00803]	
[PRS_E2E_00804]	
[PRS_E2E_00805]	
[PRS_E2E_00806]	
[PRS_E2E_00807]	
[PRS_E2E_00808]	
[PRS_E2E_00809]	

▽

△

Number	Heading
[PRS_E2E_00810]	
[PRS_E2E_00811]	Draft
[PRS_E2E_00812]	Draft
[PRS_E2E_00813]	Draft
[PRS_E2E_00814]	Draft
[PRS_E2E_00815]	
[PRS_E2E_00816]	
[PRS_E2E_00817]	
[PRS_E2E_00818]	
[PRS_E2E_00819]	
[PRS_E2E_00820]	
[PRS_E2E_00821]	
[PRS_E2E_00822]	
[PRS_E2E_00823]	
[PRS_E2E_00824]	
[PRS_E2E_00825]	
[PRS_E2E_00826]	Mapping Profile 4m to State-Machine
[PRS_E2E_00827]	Mapping Profile 7m to State-Machine
[PRS_E2E_00828]	
[PRS_E2E_00829]	
[PRS_E2E_00830]	
[PRS_E2E_00831]	
[PRS_E2E_00832]	
[PRS_E2E_00834]	
[PRS_E2E_00835]	
[PRS_E2E_00836]	
[PRS_E2E_00837]	
[PRS_E2E_00840]	
[PRS_E2E_00841]	
[PRS_E2E_00842]	
[PRS_E2E_00843]	
[PRS_E2E_00844]	
[PRS_E2E_00848]	
[PRS_E2E_00849]	
[PRS_E2E_00850]	Mapping Profile 8 to State Machine
[PRS_E2E_UC_00055]	

▽

△

Number	Heading
[PRS_E2E_UC_00062]	
[PRS_E2E_UC_00063]	
[PRS_E2E_UC_00071]	
[PRS_E2E_UC_00072]	
[PRS_E2E_UC_00073]	
[PRS_E2E_UC_00171]	
[PRS_E2E_UC_00173]	
[PRS_E2E_UC_00235]	
[PRS_E2E_UC_00238]	
[PRS_E2E_UC_00239]	
[PRS_E2E_UC_00308]	
[PRS_E2E_UC_00317]	
[PRS_E2E_UC_00320]	
[PRS_E2E_UC_00321]	
[PRS_E2E_UC_00325]	
[PRS_E2E_UC_00328]	
[PRS_E2E_UC_00606]	
[PRS_E2E_UC_00743]	
[PRS_E2E_UC_00754]	
[PRS_E2E_UC_00786]	
[PRS_E2E_UC_00797]	

▽

△

Number	Heading
[PRS_E2E_USE_00741]	

Table B.2: Added Specification Items in R20-11

B.2.5 Changed Specification Items in R20-11

Number	Heading
[PRS_E2E_00070]	
[PRS_E2E_00082]	
[PRS_E2E_00117]	
[PRS_E2E_00118]	
[PRS_E2E_00119]	
[PRS_E2E_00120]	
[PRS_E2E_00125]	
[PRS_E2E_00126]	
[PRS_E2E_00134]	
[PRS_E2E_00150]	
[PRS_E2E_00195]	
[PRS_E2E_00218]	
[PRS_E2E_00219]	
[PRS_E2E_00299]	
[PRS_E2E_00318]	
[PRS_E2E_00319]	
[PRS_E2E_00322]	
[PRS_E2E_00323]	
[PRS_E2E_00324]	
[PRS_E2E_00330]	
[PRS_E2E_00345]	
[PRS_E2E_00372]	
[PRS_E2E_00394]	
[PRS_E2E_00401]	
[PRS_E2E_00421]	
[PRS_E2E_00479]	
[PRS_E2E_00480]	
[PRS_E2E_00484]	
[PRS_E2E_00485]	
[PRS_E2E_00503]	

▽

△

Number	Heading
[PRS_E2E_00522]	
[PRS_E2E_00527]	
[PRS_E2E_00588]	
[PRS_E2E_00589]	
[PRS_E2E_00590]	
[PRS_E2E_00591]	
[PRS_E2E_00592]	
[PRS_E2E_00593]	
[PRS_E2E_00594]	
[PRS_E2E_00595]	
[PRS_E2E_00598]	Mapping Profile 1 to State Machine
[PRS_E2E_00599]	Mapping Profile 2 to State Machine
[PRS_E2E_00600]	Mapping Profile 4 to State Machine
[PRS_E2E_00601]	Mapping Profile 5 to State Machine
[PRS_E2E_00602]	Mapping Profile 6 to State Machine
[PRS_E2E_00603]	Mapping Profile 7 to State Machine
[PRS_E2E_00604]	Mapping Profile 11 to State Machine
[PRS_E2E_00605]	Mapping Profile 22 to State Machine
[PRS_E2E_00608]	
[PRS_E2E_00609]	
[PRS_E2E_00610]	
[PRS_E2E_00611]	
[PRS_E2E_00612]	
[PRS_E2E_00613]	
[PRS_E2E_00614]	
[PRS_E2E_00615]	
[PRS_E2E_00616]	
[PRS_E2E_00617]	
[PRS_E2E_00618]	
[PRS_E2E_00619]	
[PRS_E2E_00620]	
[PRS_E2E_00621]	
[PRS_E2E_00622]	
[PRS_E2E_00623]	
[PRS_E2E_00624]	
[PRS_E2E_00625]	
[PRS_E2E_00626]	
[PRS_E2E_00627]	

▽

△

Number	Heading
[PRS_E2E_00628]	
[PRS_E2E_00629]	
[PRS_E2E_00630]	
[PRS_E2E_00631]	
[PRS_E2E_00632]	
[PRS_E2E_00633]	
[PRS_E2E_00634]	
[PRS_E2E_00635]	
[PRS_E2E_00636]	
[PRS_E2E_00637]	
[PRS_E2E_00638]	
[PRS_E2E_00639]	
[PRS_E2E_UC_00051]	
[PRS_E2E_UC_00061]	
[PRS_E2E_UC_00170]	
[PRS_E2E_UC_00316]	
[PRS_E2E_UC_00351]	
[PRS_E2E_UC_00466]	

Table B.3: Changed Specification Items in R20-11

B.2.6 Deleted Specification Items in R20-11

Number	Heading
[PRS_E2E_00217]	
[PRS_E2E_00221]	
[PRS_E2E_00227]	
[PRS_E2E_00228]	
[PRS_E2E_00307]	
[PRS_E2E_00584]	
[PRS_E2E_00585]	
[PRS_E2E_00586]	
[PRS_E2E_00587]	

▽

△

Number	Heading
[PRS_E2E_UC_00237]	
[PRS_E2E_USE_00235]	
[PRS_E2E_USE_00236]	
[PRS_E2E_USE_00237]	
[PRS_E2E_USE_00321]	
[PRS_E2E_USE_00325]	
[PRS_E2E_USE_00606]	

Table B.4: Deleted Specification Items in R20-11

B.3 Traceable item history of this document according to AUTOSAR Release R21-11

B.3.1 Added Constraints in R21-11

none

B.3.2 Changed Constraints in R21-11

none

B.3.3 Deleted Constraints in R21-11

Number	Heading
[constr_3176]	Value range of `windowSizeValid`
[constr_3177]	Dependency between `maxErrorStateValid`, `maxErrorStateInit` and `maxErrorStateInvalid`
[constr_3178]	Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid`
[constr_3179]	Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeValid`
[constr_3180]	Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSizeValid`

▽

△

Number	Heading
[constr_3181]	Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeValid`
[constr_6300]	MaxDeltaCounter for Client-Server Communication
[constr_6301]	Dependency between `windowSizeInvalid` and `windowSizeValid`
[constr_6302]	Dependency between `windowSizeInit` and `windowSizeValid`

Table B.5: Deleted Constraints in R21-11

B.3.4 Added Specification Items in R21-11

Number	Heading
[PRS_E2E_00507]	
[PRS_E2E_01107]	
[PRS_E2E_01154]	
[PRS_E2E_01155]	
[PRS_E2E_01156]	
[PRS_E2E_01157]	
[PRS_E2E_01159]	
[PRS_E2E_01160]	
[PRS_E2E_01161]	
[PRS_E2E_01162]	
[PRS_E2E_01163]	
[PRS_E2E_01164]	
[PRS_E2E_01165]	
[PRS_E2E_01166]	
[PRS_E2E_01167]	
[PRS_E2E_01169]	
[PRS_E2E_01170]	
[PRS_E2E_01171]	
[PRS_E2E_01172]	
[PRS_E2E_01173]	
[PRS_E2E_01174]	
[PRS_E2E_01175]	
[PRS_E2E_01176]	
[PRS_E2E_01177]	
[PRS_E2E_01178]	
[PRS_E2E_01179]	
[PRS_E2E_01180]	

▽

△

Number	Heading
[PRS_E2E_01181]	Draft
[PRS_E2E_01182]	Draft
[PRS_E2E_01183]	Draft
[PRS_E2E_01184]	Draft
[PRS_E2E_01185]	
[PRS_E2E_01186]	
[PRS_E2E_01187]	
[PRS_E2E_01188]	
[PRS_E2E_01189]	
[PRS_E2E_01190]	
[PRS_E2E_01191]	
[PRS_E2E_01192]	
[PRS_E2E_01193]	
[PRS_E2E_01194]	
[PRS_E2E_01195]	
[PRS_E2E_01196]	
[PRS_E2E_01197]	
[PRS_E2E_01198]	
[PRS_E2E_01199]	
[PRS_E2E_01200]	
[PRS_E2E_01201]	
[PRS_E2E_01202]	
[PRS_E2E_01203]	
[PRS_E2E_02355]	
[PRS_E2E_02356]	
[PRS_E2E_02357]	
[PRS_E2E_02358]	
[PRS_E2E_02359]	
[PRS_E2E_02360]	
[PRS_E2E_02361]	
[PRS_E2E_02362]	
[PRS_E2E_02363]	
[PRS_E2E_02364]	
[PRS_E2E_02365]	
[PRS_E2E_02366]	
[PRS_E2E_02367]	
[PRS_E2E_02368]	
[PRS_E2E_02369]	

▽

△

Number	Heading
[PRS_E2E_02376]	
[PRS_E2E_02615]	
[PRS_E2E_02616]	
[PRS_E2E_02617]	
[PRS_E2E_02618]	
[PRS_E2E_CONSTR_03176]	Value range of `windowSizeValid`
[PRS_E2E_CONSTR_03177]	Dependency between `maxErrorStateValid`
[PRS_E2E_CONSTR_03178]	Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid`
[PRS_E2E_CONSTR_03179]	Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeInit`
[PRS_E2E_CONSTR_03180]	Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSizeValid`
[PRS_E2E_CONSTR_03181]	Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeInvalid`
[PRS_E2E_CONSTR_06300]	MaxDeltaCounter for Client-Server Communication
[PRS_E2E_CONSTR_06301]	Dependency between `windowSizeInvalid` and `windowSizeValid`
[PRS_E2E_CONSTR_06302]	Dependency between `windowSizeInit` and `windowSizeValid`
[PRS_E2E_UC_01158]	
[PRS_E2E_UC_01168]	

Table B.6: Added Specification Items in R21-11

B.3.5 Changed Specification Items in R21-11

Number	Heading
[PRS_E2E_00329]	
[PRS_E2E_00355]	
[PRS_E2E_00356]	
[PRS_E2E_00357]	
[PRS_E2E_00358]	
[PRS_E2E_00359]	
[PRS_E2E_00360]	

▽

△

Number	Heading
[PRS_E2E_00361]	
[PRS_E2E_00362]	
[PRS_E2E_00363]	
[PRS_E2E_00364]	
[PRS_E2E_00365]	
[PRS_E2E_00366]	
[PRS_E2E_00367]	
[PRS_E2E_00368]	
[PRS_E2E_00369]	
[PRS_E2E_00372]	
[PRS_E2E_00376]	
[PRS_E2E_00394]	
[PRS_E2E_00404]	
[PRS_E2E_00405]	
[PRS_E2E_00407]	
[PRS_E2E_00409]	
[PRS_E2E_00412]	
[PRS_E2E_00413]	
[PRS_E2E_00414]	
[PRS_E2E_00416]	
[PRS_E2E_00424]	
[PRS_E2E_00425]	
[PRS_E2E_00426]	
[PRS_E2E_00428]	
[PRS_E2E_00429]	
[PRS_E2E_00431]	
[PRS_E2E_00432]	
[PRS_E2E_00433]	
[PRS_E2E_00434]	
[PRS_E2E_00436]	
[PRS_E2E_00469]	
[PRS_E2E_00470]	
[PRS_E2E_00479]	
[PRS_E2E_00486]	
[PRS_E2E_00487]	
[PRS_E2E_00488]	
[PRS_E2E_00489]	
[PRS_E2E_00490]	

▽

△

Number	Heading
[PRS_E2E_00491]	
[PRS_E2E_00492]	
[PRS_E2E_00493]	
[PRS_E2E_00494]	
[PRS_E2E_00495]	
[PRS_E2E_00496]	
[PRS_E2E_00497]	
[PRS_E2E_00498]	
[PRS_E2E_00499]	
[PRS_E2E_00500]	
[PRS_E2E_00501]	
[PRS_E2E_00503]	
[PRS_E2E_00522]	
[PRS_E2E_00615]	
[PRS_E2E_00616]	
[PRS_E2E_00617]	
[PRS_E2E_00618]	
[PRS_E2E_00619]	
[PRS_E2E_00620]	
[PRS_E2E_00623]	
[PRS_E2E_00624]	
[PRS_E2E_00626]	
[PRS_E2E_00627]	
[PRS_E2E_00628]	
[PRS_E2E_00629]	
[PRS_E2E_00646]	
[PRS_E2E_00648]	
[PRS_E2E_00651]	
[PRS_E2E_00654]	
[PRS_E2E_00657]	
[PRS_E2E_00660]	
[PRS_E2E_00663]	
[PRS_E2E_00666]	
[PRS_E2E_00683]	
[PRS_E2E_00684]	
[PRS_E2E_00685]	
[PRS_E2E_00686]	
[PRS_E2E_00687]	

▽

△

Number	Heading
[PRS_E2E_00688]	
[PRS_E2E_00689]	
[PRS_E2E_00690]	
[PRS_E2E_00691]	
[PRS_E2E_00692]	
[PRS_E2E_00693]	
[PRS_E2E_00694]	
[PRS_E2E_00695]	
[PRS_E2E_00696]	
[PRS_E2E_00697]	
[PRS_E2E_00698]	
[PRS_E2E_00699]	
[PRS_E2E_00700]	
[PRS_E2E_00701]	
[PRS_E2E_00702]	
[PRS_E2E_00706]	
[PRS_E2E_00712]	
[PRS_E2E_00713]	
[PRS_E2E_00714]	
[PRS_E2E_00715]	
[PRS_E2E_00716]	
[PRS_E2E_00717]	
[PRS_E2E_00718]	
[PRS_E2E_00719]	
[PRS_E2E_00720]	
[PRS_E2E_00721]	
[PRS_E2E_00722]	
[PRS_E2E_00723]	
[PRS_E2E_00724]	
[PRS_E2E_00725]	
[PRS_E2E_00726]	
[PRS_E2E_00727]	
[PRS_E2E_00728]	
[PRS_E2E_00729]	
[PRS_E2E_00730]	
[PRS_E2E_00731]	
[PRS_E2E_00735]	
[PRS_E2E_00736]	

▽

△

Number	Heading
[PRS_E2E_00824]	
[PRS_E2E_00831]	
[PRS_E2E_00836]	
[PRS_E2E_00840]	
[PRS_E2E_UC_00055]	
[PRS_E2E_UC_00071]	
[PRS_E2E_UC_00072]	
[PRS_E2E_UC_00073]	
[PRS_E2E_UC_00170]	
[PRS_E2E_UC_00171]	
[PRS_E2E_UC_00173]	
[PRS_E2E_UC_00235]	
[PRS_E2E_UC_00236]	
[PRS_E2E_UC_00238]	
[PRS_E2E_UC_00239]	
[PRS_E2E_UC_00308]	
[PRS_E2E_UC_00316]	
[PRS_E2E_UC_00317]	
[PRS_E2E_UC_00320]	
[PRS_E2E_UC_00321]	
[PRS_E2E_UC_00325]	
[PRS_E2E_UC_00327]	
[PRS_E2E_UC_00328]	

▽

△

Number	Heading
[PRS_E2E_UC_00351]	
[PRS_E2E_UC_00463]	
[PRS_E2E_UC_00464]	
[PRS_E2E_UC_00606]	
[PRS_E2E_UC_00743]	
[PRS_E2E_UC_00754]	
[PRS_E2E_UC_00786]	
[PRS_E2E_UC_00797]	
[PRS_E2E_USE_00741]	

Table B.7: Changed Specification Items in R21-11

B.3.6 Deleted Specification Items in R21-11

Number	Heading
[PRS_E2E_0507]	

Table B.8: Deleted Specification Items in R21-11

B.4 Traceable item history of this document according to AUTOSAR Release R22-11

B.4.1 Added Constraints in R22-11

none

B.4.2 Changed Constraints in R22-11

none

B.4.3 Deleted Constraints in R22-11

none

B.4.4 Added Specification Items in R22-11

Number	Heading
[PRS_E2E_00851]	
[PRS_E2E_00852]	
[PRS_E2E_01205]	
[PRS_E2E_01206]	
[PRS_E2E_01207]	
[PRS_E2E_01209]	
[PRS_E2E_01210]	
[PRS_E2E_01211]	
[PRS_E2E_01212]	
[PRS_E2E_01213]	
[PRS_E2E_01214]	
[PRS_E2E_01215]	
[PRS_E2E_01216]	
[PRS_E2E_01217]	
[PRS_E2E_01218]	
[PRS_E2E_01219]	
[PRS_E2E_01220]	
[PRS_E2E_01221]	
[PRS_E2E_01222]	
[PRS_E2E_01223]	
[PRS_E2E_01224]	
[PRS_E2E_01225]	
[PRS_E2E_01226]	
[PRS_E2E_01227]	
[PRS_E2E_01228]	
[PRS_E2E_01250]	
[PRS_E2E_01251]	
[PRS_E2E_01252]	
[PRS_E2E_CONSTR_06303]	MaxDeltaCounter for Client-Server Communication (client)
[PRS_E2E_UC_01204]	

Table B.9: Added Specification Items in R22-11

B.4.5 Changed Specification Items in R22-11

Number	Heading
[PRS_E2E_00404]	
[PRS_E2E_00405]	
[PRS_E2E_00407]	
[PRS_E2E_00409]	
[PRS_E2E_00412]	
[PRS_E2E_00413]	
[PRS_E2E_00414]	
[PRS_E2E_00416]	
[PRS_E2E_00424]	
[PRS_E2E_00425]	
[PRS_E2E_00426]	
[PRS_E2E_00428]	
[PRS_E2E_00429]	
[PRS_E2E_00431]	
[PRS_E2E_00432]	
[PRS_E2E_00433]	
[PRS_E2E_00434]	
[PRS_E2E_00436]	
[PRS_E2E_00469]	
[PRS_E2E_00470]	
[PRS_E2E_01171]	
[PRS_E2E_01178]	
[PRS_E2E_01185]	
[PRS_E2E_01186]	
[PRS_E2E_01187]	
[PRS_E2E_01188]	
[PRS_E2E_01189]	
[PRS_E2E_01190]	
[PRS_E2E_01191]	
[PRS_E2E_01192]	
[PRS_E2E_01193]	
[PRS_E2E_01194]	
[PRS_E2E_01195]	
[PRS_E2E_01196]	
[PRS_E2E_01197]	
[PRS_E2E_01198]	

△

Number	Heading
[PRS_E2E_01200]	
[PRS_E2E_01201]	
[PRS_E2E_CONSTR_06300]	MaxDeltaCounter for Client-Server Communication (server)

Table B.10: Changed Specification Items in R22-11

B.4.6 Deleted Specification Items in R22-11

Number	Heading
[PRS_E2E_00326]	
[PRS_E2E_00329]	
[PRS_E2E_00330]	
[PRS_E2E_00355]	
[PRS_E2E_00356]	
[PRS_E2E_00357]	
[PRS_E2E_00358]	
[PRS_E2E_00359]	
[PRS_E2E_00360]	
[PRS_E2E_00361]	
[PRS_E2E_00362]	
[PRS_E2E_00363]	
[PRS_E2E_00364]	
[PRS_E2E_00365]	
[PRS_E2E_00366]	
[PRS_E2E_00367]	
[PRS_E2E_00368]	
[PRS_E2E_00369]	
[PRS_E2E_00376]	
[PRS_E2E_00478]	
[PRS_E2E_00481]	
[PRS_E2E_00482]	
[PRS_E2E_00483]	
[PRS_E2E_00484]	
[PRS_E2E_00485]	
[PRS_E2E_00486]	
[PRS_E2E_00487]	
[PRS_E2E_00488]	

▽

△

Number	Heading
[PRS_E2E_00489]	
[PRS_E2E_00490]	
[PRS_E2E_00491]	
[PRS_E2E_00492]	
[PRS_E2E_00493]	
[PRS_E2E_00494]	
[PRS_E2E_00495]	
[PRS_E2E_00496]	
[PRS_E2E_00497]	
[PRS_E2E_00498]	
[PRS_E2E_00499]	
[PRS_E2E_00500]	
[PRS_E2E_00501]	
[PRS_E2E_00590]	
[PRS_E2E_00593]	
[PRS_E2E_00615]	
[PRS_E2E_00616]	
[PRS_E2E_00617]	
[PRS_E2E_00618]	
[PRS_E2E_00626]	
[PRS_E2E_00627]	
[PRS_E2E_00628]	
[PRS_E2E_00629]	
[PRS_E2E_00649]	
[PRS_E2E_00650]	
[PRS_E2E_00658]	
[PRS_E2E_00659]	
[PRS_E2E_00679]	
[PRS_E2E_00680]	
[PRS_E2E_00681]	
[PRS_E2E_00682]	
[PRS_E2E_00683]	
[PRS_E2E_00684]	
[PRS_E2E_00685]	
[PRS_E2E_00686]	
[PRS_E2E_00687]	
[PRS_E2E_00688]	
[PRS_E2E_00689]	

▽

△

Number	Heading
[PRS_E2E_00690]	
[PRS_E2E_00691]	
[PRS_E2E_00692]	
[PRS_E2E_00693]	
[PRS_E2E_00694]	
[PRS_E2E_00695]	
[PRS_E2E_00696]	
[PRS_E2E_00697]	
[PRS_E2E_00698]	
[PRS_E2E_00699]	
[PRS_E2E_00700]	
[PRS_E2E_00701]	
[PRS_E2E_00702]	
[PRS_E2E_00703]	
[PRS_E2E_00704]	
[PRS_E2E_00705]	
[PRS_E2E_00708]	
[PRS_E2E_00709]	
[PRS_E2E_00710]	
[PRS_E2E_00711]	
[PRS_E2E_00712]	
[PRS_E2E_00713]	
[PRS_E2E_00714]	
[PRS_E2E_00715]	
[PRS_E2E_00716]	
[PRS_E2E_00717]	
[PRS_E2E_00718]	
[PRS_E2E_00719]	
[PRS_E2E_00720]	
[PRS_E2E_00721]	
[PRS_E2E_00722]	
[PRS_E2E_00723]	
[PRS_E2E_00724]	
[PRS_E2E_00725]	
[PRS_E2E_00726]	
[PRS_E2E_00727]	
[PRS_E2E_00728]	
[PRS_E2E_00729]	

▽

△

Number	Heading
[PRS_E2E_00730]	
[PRS_E2E_00731]	
[PRS_E2E_00732]	
[PRS_E2E_00733]	
[PRS_E2E_00734]	
[PRS_E2E_00737]	
[PRS_E2E_00741]	
[PRS_E2E_00742]	
[PRS_E2E_00744]	
[PRS_E2E_00745]	
[PRS_E2E_00746]	
[PRS_E2E_00747]	
[PRS_E2E_00748]	
[PRS_E2E_00749]	
[PRS_E2E_00750]	
[PRS_E2E_00751]	
[PRS_E2E_00752]	
[PRS_E2E_00753]	
[PRS_E2E_00756]	
[PRS_E2E_00757]	
[PRS_E2E_00758]	
[PRS_E2E_00759]	
[PRS_E2E_00760]	
[PRS_E2E_00761]	
[PRS_E2E_00762]	
[PRS_E2E_00763]	
[PRS_E2E_00764]	
[PRS_E2E_00765]	
[PRS_E2E_00766]	
[PRS_E2E_00767]	
[PRS_E2E_00768]	Draft
[PRS_E2E_00769]	Draft
[PRS_E2E_00770]	Draft
[PRS_E2E_00771]	Draft
[PRS_E2E_00772]	
[PRS_E2E_00773]	
[PRS_E2E_00774]	
[PRS_E2E_00775]	

▽

△

Number	Heading
[PRS_E2E_00776]	
[PRS_E2E_00777]	
[PRS_E2E_00778]	
[PRS_E2E_00779]	
[PRS_E2E_00780]	
[PRS_E2E_00781]	
[PRS_E2E_00784]	
[PRS_E2E_00785]	
[PRS_E2E_00787]	
[PRS_E2E_00788]	
[PRS_E2E_00789]	
[PRS_E2E_00790]	
[PRS_E2E_00791]	
[PRS_E2E_00792]	
[PRS_E2E_00793]	
[PRS_E2E_00794]	
[PRS_E2E_00795]	
[PRS_E2E_00796]	
[PRS_E2E_00799]	
[PRS_E2E_00800]	
[PRS_E2E_00801]	
[PRS_E2E_00802]	
[PRS_E2E_00803]	
[PRS_E2E_00804]	
[PRS_E2E_00805]	
[PRS_E2E_00806]	
[PRS_E2E_00807]	
[PRS_E2E_00808]	
[PRS_E2E_00809]	
[PRS_E2E_00810]	
[PRS_E2E_00811]	Draft
[PRS_E2E_00812]	Draft
[PRS_E2E_00813]	Draft
[PRS_E2E_00814]	Draft
[PRS_E2E_00815]	
[PRS_E2E_00816]	
[PRS_E2E_00817]	
[PRS_E2E_00818]	

▽

△

Number	Heading
[PRS_E2E_00819]	
[PRS_E2E_00820]	
[PRS_E2E_00821]	
[PRS_E2E_00822]	
[PRS_E2E_00823]	
[PRS_E2E_00824]	
[PRS_E2E_00825]	
[PRS_E2E_00829]	
[PRS_E2E_00830]	
[PRS_E2E_00831]	
[PRS_E2E_00832]	
[PRS_E2E_00834]	
[PRS_E2E_00835]	
[PRS_E2E_00836]	
[PRS_E2E_00837]	
[PRS_E2E_00840]	
[PRS_E2E_00841]	
[PRS_E2E_00842]	
[PRS_E2E_00843]	
[PRS_E2E_00844]	
[PRS_E2E_00848]	
[PRS_E2E_00849]	
[PRS_E2E_02355]	
[PRS_E2E_02356]	
[PRS_E2E_02357]	
[PRS_E2E_02358]	
[PRS_E2E_02359]	
[PRS_E2E_02360]	
[PRS_E2E_02361]	
[PRS_E2E_02362]	
[PRS_E2E_02363]	
[PRS_E2E_02364]	
[PRS_E2E_02365]	
[PRS_E2E_02366]	
[PRS_E2E_02367]	
[PRS_E2E_02368]	
[PRS_E2E_02369]	
[PRS_E2E_02376]	

▽

△

Number	Heading
[PRS_E2E_02615]	
[PRS_E2E_02616]	
[PRS_E2E_02617]	
[PRS_E2E_02618]	
[PRS_E2E_UC_00327]	
[PRS_E2E_UC_00328]	
[PRS_E2E_UC_00743]	
[PRS_E2E_UC_00754]	
[PRS_E2E_UC_00786]	
[PRS_E2E_UC_00797]	

Table B.11: Deleted Specification Items in R22-11

B.5 Traceable item history of this document according to AUTOSAR Release R23-11

B.5.1 Added Constraints in R23-11

none

B.5.2 Changed Constraints in R23-11

Number	Heading
[PRS_E2E_CONSTR_03179]	Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeInit`
[PRS_E2E_CONSTR_03181]	Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeInvalid`
[PRS_E2E_CONSTR_06301]	Dependency between `windowSizeInvalid` and `windowSizeValid`

▽

△

Number	Heading
[PRS_E2E_CONSTR_06302]	Dependency between `windowSizeInit` and `windowSizeValid`

Table B.12: Changed Constraints in R23-11

B.5.3 Deleted Constraints in R23-11

none

B.5.4 Added Specification Items in R23-11

Number	Heading
[PRS_E2E_00853]	

Table B.13: Added Specification Items in R23-11

B.5.5 Changed Specification Items in R23-11

Number	Heading
[PRS_E2E_00121]	
[PRS_E2E_00322]	
[PRS_E2E_00324]	
[PRS_E2E_00345]	
[PRS_E2E_00480]	
[PRS_E2E_00600]	Mapping Profile 4,5,6,7,8,11,22,44,4m,7m,8m,44m to State Machine
[PRS_E2E_00607]	
[PRS_E2E_00675]	
[PRS_E2E_00676]	
[PRS_E2E_00677]	

Table B.14: Changed Specification Items in R23-11

B.5.6 Deleted Specification Items in R23-11

Number	Heading
[PRS_E2E_00601]	Mapping Profile 5 to State Machine
[PRS_E2E_00602]	Mapping Profile 6 to State Machine
[PRS_E2E_00603]	Mapping Profile 7 to State Machine
[PRS_E2E_00604]	Mapping Profile 11 to State Machine
[PRS_E2E_00605]	Mapping Profile 22 to State Machine
[PRS_E2E_00738]	Mapping Profile 44 to State Machine
[PRS_E2E_00826]	Mapping Profile 4m to State-Machine
[PRS_E2E_00827]	Mapping Profile 7m to State-Machine
[PRS_E2E_00850]	Mapping Profile 8 to State Machine

Table B.15: Deleted Specification Items in R23-11

B.6 Traceable item history of this document according to AUTOSAR Release R24-11

B.6.1 Added Constraints in R24-11

none

B.6.2 Changed Constraints in R24-11

none

B.6.3 Deleted Constraints in R24-11

none

B.6.4 Added Specification Items in R24-11

Number	Heading
[PRS_E2E_00855]	Error Codes
[PRS_E2E_00856]	E2E Profile Xm Message Type Enumeration
[PRS_E2E_00857]	E2E Profile Xm Message Result Enumeration
[PRS_E2E_00858]	E2E Profile Protect State Type
[PRS_E2E_00859]	E2E Profile XX Check State Type

▽

△

Number	Heading
[PRS_E2E_00860]	E2E Profile Check Status Enumeration
[PRS_E2E_00862]	E2E Profile XXm Protect State Type
[PRS_E2E_00863]	E2E Profile XXm Check State Type
[PRS_E2E_00864]	E2E Profile XXm Check Status Enumeration
[PRS_E2E_00865]	E2E Profile 1 mechanisms
[PRS_E2E_00866]	Profile 1 Protect State Type
[PRS_E2E_00867]	E2E Profile 1 Check Status Type Members
[PRS_E2E_00868]	E2E Profile 1 Check Status Enumeration
[PRS_E2E_00869]	E2E Profile 1 Configuration Type
[PRS_E2E_00870]	E2E Profile 2 mechanisms
[PRS_E2E_00871]	E2E Profile 2 Protect State Type
[PRS_E2E_00872]	E2E Profile 2 Check Status Type
[PRS_E2E_00873]	E2E Profile 2 Check Status Enumeration
[PRS_E2E_00874]	E2E Profile 2 Configuration Type
[PRS_E2E_00875]	E2E Profile 4 mechanisms
[PRS_E2E_00876]	E2E Profile 4 Configuration Type
[PRS_E2E_00877]	E2E Profile 5 mechanisms
[PRS_E2E_00878]	E2E Profile 5 Protect State Type
[PRS_E2E_00879]	E2E Profile 5 Check Status Type
[PRS_E2E_00880]	E2E Profile 5 Check Status Enumeration
[PRS_E2E_00881]	E2E Profile 5 Configuration Type
[PRS_E2E_00882]	E2E Profile 6 mechanisms
[PRS_E2E_00883]	E2E Profile 6 Protect State Type
[PRS_E2E_00884]	E2E Profile 6 Check Status Type
[PRS_E2E_00885]	E2E Profile 6 Check Status Enumeration
[PRS_E2E_00886]	E2E Profile 6 Configuration Type
[PRS_E2E_00887]	E2E Profile 7 Configuration Type
[PRS_E2E_00888]	E2E Profile 11 mechanisms
[PRS_E2E_00889]	E2E Profile 11 Protect State Type
[PRS_E2E_00891]	E2E Profile 11 Check Status Type
[PRS_E2E_00892]	E2E Profile 11 Check Status Enumeration
[PRS_E2E_00893]	E2E Profile 11 Configuration Type
[PRS_E2E_00894]	E2E Profile 22 mechanisms
[PRS_E2E_00898]	E2E Profile 22 Configuration Type
[PRS_E2E_00899]	E2E Profile 4m mechanisms
[PRS_E2E_00900]	E2E Profile 4m Configuration Type
[PRS_E2E_00901]	E2E Profile 8 mechanisms
[PRS_E2E_00902]	E2E Profile 8 Configuration Type

▽

△

Number	Heading
[PRS_E2E_00903]	E2E Profile 7m mechanisms
[PRS_E2E_00904]	E2E Profile 7 mechanisms
[PRS_E2E_00905]	E2E Profile 7m Configuration Type
[PRS_E2E_00906]	E2E Profile 44 mechanisms
[PRS_E2E_00907]	E2E Profile 44 Configuration Type
[PRS_E2E_00908]	E2E Profile 8m mechanisms
[PRS_E2E_00909]	E2E Profile 8m Configuration Type
[PRS_E2E_00910]	E2E Profile 44m mechanisms
[PRS_E2E_00911]	E2E Profile 44m Configuration Type
[PRS_E2E_00912]	E2E State Machine Configuration Type
[PRS_E2E_00913]	E2E State Machine State Type
[PRS_E2E_00914]	E2E State Machine Check Status Enumeration
[PRS_E2E_00915]	E2E State Machine Check Status Enumeration
[PRS_E2E_00916]	E2E Profile 1 specific Check Status Mapping since R4.2
[PRS_E2E_00917]	E2E Profile 1 specific Check Status Mapping prior to R4.2
[PRS_E2E_00918]	E2E Profile 2 specific Check Status Mapping since R4.2
[PRS_E2E_00919]	E2E Profile 2 specific Check Status Mapping prior to R4.2
[PRS_E2E_00920]	E2E Profile specific Check Status Mapping
[PRS_E2E_00921]	Mapping between profile independent and specific states
[PRS_E2E_00922]	Mapping between Communication Channel and State Machine States
[PRS_E2E_00923]	Disable/enable configuration parameters
[PRS_E2E_00924]	E2E profile specific configuration parameters
[PRS_E2E_00925]	E2E state machine configuration parameters
[PRS_E2E_01253]	Compute Offset of Protect Function
[PRS_E2E_01254]	Compute CRC - Data ID (Protect and Check Function)
[PRS_E2E_01255]	Compute CRC - Data ID (Forward Function)
[PRS_E2E_01318]	Profile 76 Mechanisms
[PRS_E2E_01400]	Profile 76 Initialization of counter
[PRS_E2E_01401]	Profile 76 CRC Calculation Algorithm
[PRS_E2E_01402]	Profile 76 CRC Calculation over E2E Header
[PRS_E2E_01403]	Profile 76 Check function
[PRS_E2E_01404]	E2E Profile 76 Check
[PRS_E2E_01405]	Profile 76 Verification of inputs of Check function
[PRS_E2E_01406]	Profile 76 Read Counter
[PRS_E2E_01407]	Profile 76 Read CRC
[PRS_E2E_01408]	Profile 76 Perform checks
[PRS_E2E_01409]	Profile 76 Protect Function
[PRS_E2E_01410]	E2E Profile 76 Protect State Type

▽

△

Number	Heading
[PRS_E2E_01411]	Profile 76 Check Function
[PRS_E2E_01412]	E2E Profile 76 Check State Type
[PRS_E2E_01413]	Profile 76 Status
[PRS_E2E_01414]	E2E Profile 76 Check Status Enumeration
[PRS_E2E_01415]	Profile 76 Configuration
[PRS_E2E_01416]	E2E Profile 76 Configuration Type
[PRS_E2E_01420]	Profile 76 Protect Function
[PRS_E2E_01421]	E2E Profile 76 Protect
[PRS_E2E_01422]	Profile 76 Verification of Inputs of Protect Function
[PRS_E2E_01424]	Profile 76 Write Counter
[PRS_E2E_01427]	Profile 76 Compute CRC
[PRS_E2E_01428]	E2E Profile 76 Protect and Check step "Compute CRC"
[PRS_E2E_01429]	Profile 76 Write CRC
[PRS_E2E_01430]	Profile 76 Increment Counter
[PRS_E2E_01431]	E2E Profile 76 Protect step "Increment Counter CRC"
[PRS_E2E_01433]	Profile 76 Compute CRC
[PRS_E2E_01434]	E2E Profile 76 Protect step "Write Counter"
[PRS_E2E_01436]	E2E Profile 76 Check
[PRS_E2E_01437]	Parameter Config->combinedNoDataInitCount

Table B.16: Added Specification Items in R24-11

B.6.5 Changed Specification Items in R24-11

Number	Heading
[PRS_E2E_00218]	
[PRS_E2E_00219]	
[PRS_E2E_00322]	
[PRS_E2E_00324]	
[PRS_E2E_00345]	
[PRS_E2E_00372]	
[PRS_E2E_00394]	
[PRS_E2E_00479]	
[PRS_E2E_00480]	
[PRS_E2E_00503]	
[PRS_E2E_00522]	
[PRS_E2E_00531]	

▽

△

Number	Heading
[PRS_E2E_00533]	
[PRS_E2E_00588]	
[PRS_E2E_00589]	
[PRS_E2E_00591]	
[PRS_E2E_00592]	
[PRS_E2E_00594]	
[PRS_E2E_00596]	
[PRS_E2E_00597]	
[PRS_E2E_00598]	Mapping Profile 1 to State Machine
[PRS_E2E_00599]	Mapping Profile 2 to State Machine
[PRS_E2E_00600]	Mapping Profile 4,5,6,7,8,11,22,44,4m,7m,8m,44m to State Machine
[PRS_E2E_00637]	
[PRS_E2E_00641]	Mapping Profile 1 to State Machine
[PRS_E2E_00644]	
[PRS_E2E_00645]	
[PRS_E2E_00646]	
[PRS_E2E_00647]	
[PRS_E2E_00648]	
[PRS_E2E_00651]	
[PRS_E2E_00652]	
[PRS_E2E_00653]	
[PRS_E2E_00654]	
[PRS_E2E_00655]	
[PRS_E2E_00656]	
[PRS_E2E_00657]	
[PRS_E2E_00660]	
[PRS_E2E_00661]	
[PRS_E2E_00662]	
[PRS_E2E_00663]	
[PRS_E2E_00666]	
[PRS_E2E_00667]	
[PRS_E2E_00668]	
[PRS_E2E_00669]	E2E State Machine State Type
[PRS_E2E_00670]	Mapping Profile 2 to State Machine
[PRS_E2E_00673]	
[PRS_E2E_00675]	
[PRS_E2E_00676]	
[PRS_E2E_00677]	

▽

△

Number	Heading
[PRS_E2E_00678]	
[PRS_E2E_00706]	
[PRS_E2E_00707]	
[PRS_E2E_00735]	
[PRS_E2E_00736]	
[PRS_E2E_00739]	
[PRS_E2E_00740]	
[PRS_E2E_00743]	
[PRS_E2E_00783]	
[PRS_E2E_00851]	
[PRS_E2E_00852]	
[PRS_E2E_01107]	
[PRS_E2E_01154]	
[PRS_E2E_01155]	
[PRS_E2E_01199]	
[PRS_E2E_01200]	
[PRS_E2E_01201]	
[PRS_E2E_01202]	
[PRS_E2E_01250]	
[PRS_E2E_01251]	
[PRS_E2E_01252]	

Table B.17: Changed Specification Items in R24-11

B.6.6 Deleted Specification Items in R24-11

Number	Heading
[PRS_E2E_00595]	
[PRS_E2E_00638]	
[PRS_E2E_00664]	
[PRS_E2E_00665]	

Table B.18: Deleted Specification Items in R24-11

B.7 Traceable item history of this document according to AUTOSAR Release R25-11

B.7.1 Added Constraints in R25-11

none

B.7.2 Changed Constraints in R25-11

none

B.7.3 Deleted Constraints in R25-11

Number	Heading
[PRS_E2E_CONSTR_03176]	Value range of `windowSizeValid`
[PRS_E2E_CONSTR_03177]	Dependency between `maxErrorStateValid`
[PRS_E2E_CONSTR_03178]	Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid`
[PRS_E2E_CONSTR_03179]	Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeInit`
[PRS_E2E_CONSTR_03180]	Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSizeValid`
[PRS_E2E_CONSTR_03181]	Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeInvalid`
[PRS_E2E_CONSTR_06300]	MaxDeltaCounter for Client-Server Communication (server)
[PRS_E2E_CONSTR_06301]	Dependency between `windowSizeInvalid` and `windowSizeValid`
[PRS_E2E_CONSTR_06302]	Dependency between `windowSizeInit` and `windowSizeValid`
[PRS_E2E_CONSTR_06303]	MaxDeltaCounter for Client-Server Communication (client)

Table B.19: Deleted Constraints in R25-11

B.7.4 Added Specification Items in R25-11

Number	Heading
[PRS_E2E_01438]	Uniqueness of Data IDs
[PRS_E2E_01439]	Uniqueness of Data IDs
[PRS_E2E_01440]	Uniqueness of Data IDs
[PRS_E2E_01441]	Uniqueness of Data IDs
[PRS_E2E_01442]	Uniqueness of Data IDs
[PRS_E2E_01443]	Interface to middleware
[PRS_E2E_01444]	Some/IP coverage
[PRS_E2E_01445]	E2E for methods
[PRS_E2E_01446]	Some/IP coverage
[PRS_E2E_01447]	MaxDeltaCounter for Client-Server Communication (server)
[PRS_E2E_01448]	MaxDeltaCounter for Client-Server Communication (client)
[PRS_E2E_01449]	cyclic call FTTI
[PRS_E2E_01450]	cyclic reception
[PRS_E2E_01451]	Alignment
[PRS_E2E_01452]	Alignment
[PRS_E2E_01453]	length of data
[PRS_E2E_01454]	unused areas of messages
[PRS_E2E_01455]	high nibble value at profiles 1 and 11
[PRS_E2E_01456]	Data ID values
[PRS_E2E_01457]	Value range of `windowSizeValid`
[PRS_E2E_01458]	Dependency between `windowSizeInvalid` and `windowSizeValid`
[PRS_E2E_01459]	Dependency between `windowSizeInit` and `windowSizeValid`
[PRS_E2E_01460]	Dependency between `maxErrorStateValid`
[PRS_E2E_01461]	Dependency between `minOkStateValid`, `minOkStateInit` and `minOkStateInvalid`
[PRS_E2E_01462]	Dependency between `minOkStateInit`, `maxErrorStateInit` and `windowSizeInit`
[PRS_E2E_01463]	Dependency between `minOkStateValid`, `maxErrorStateValid` and `windowSizeValid`
[PRS_E2E_01464]	Dependency between `minOkStateInvalid`, `maxErrorStateInvalid` and `windowSizeInvalid`
[PRS_E2E_01465]	transfer counter from request to response
[PRS_E2E_01466]	Data Types of PXX_GetHeaderInfo
[PRS_E2E_01467]	Data Types Description
[PRS_E2E_01468]	Data Types of PXXm_GetHeaderInfo
[PRS_E2E_01469]	Data Types Description
[PRS_E2E_01471]	PXX_GetHeaderInfo

▽

△

Number	Heading
[PRS_E2E_01472]	PXXm_GetHeaderInfo

Table B.20: Added Specification Items in R25-11

B.7.5 Changed Specification Items in R25-11

Number	Heading
[PRS_E2E_00147]	
[PRS_E2E_00876]	E2E Profile 4 Configuration Type
[PRS_E2E_00881]	E2E Profile 5 Configuration Type
[PRS_E2E_00886]	E2E Profile 6 Configuration Type
[PRS_E2E_00887]	E2E Profile 7 Configuration Type
[PRS_E2E_00900]	E2E Profile 4m Configuration Type
[PRS_E2E_00902]	E2E Profile 8 Configuration Type
[PRS_E2E_00905]	E2E Profile 7m Configuration Type
[PRS_E2E_00907]	E2E Profile 44 Configuration Type
[PRS_E2E_00909]	E2E Profile 8m Configuration Type
[PRS_E2E_00911]	E2E Profile 44m Configuration Type
[PRS_E2E_01403]	Profile 76 Check function
[PRS_E2E_01405]	Profile 76 Verification of inputs of Check function
[PRS_E2E_01406]	Profile 76 Read Counter
[PRS_E2E_01407]	Profile 76 Read CRC
[PRS_E2E_01408]	Profile 76 Perform checks
[PRS_E2E_01416]	E2E Profile 76 Configuration Type
[PRS_E2E_01420]	Profile 76 Protect Function
[PRS_E2E_01422]	Profile 76 Verification of Inputs of Protect Function
[PRS_E2E_01424]	Profile 76 Write Counter
[PRS_E2E_01427]	Profile 76 Compute CRC
[PRS_E2E_01429]	Profile 76 Write CRC
[PRS_E2E_01430]	Profile 76 Increment Counter
[PRS_E2E_01433]	Profile 76 Compute CRC

Table B.21: Changed Specification Items in R25-11

B.7.6 Deleted Specification Items in R25-11

Number	Heading
[PRS_E2E_01404]	E2E Profile 76 Check
[PRS_E2E_01421]	E2E Profile 76 Protect
[PRS_E2E_01428]	E2E Profile 76 Protect and Check step "Compute CRC"
[PRS_E2E_01431]	E2E Profile 76 Protect step "Increment Counter CRC"
[PRS_E2E_01434]	E2E Profile 76 Protect step "Write Counter"
[PRS_E2E_01436]	E2E Profile 76 Check
[PRS_E2E_UC_00051]	
[PRS_E2E_UC_00055]	
[PRS_E2E_UC_00061]	
[PRS_E2E_UC_00062]	
[PRS_E2E_UC_00063]	
[PRS_E2E_UC_00071]	
[PRS_E2E_UC_00072]	
[PRS_E2E_UC_00073]	
[PRS_E2E_UC_00170]	
[PRS_E2E_UC_00171]	
[PRS_E2E_UC_00173]	
[PRS_E2E_UC_00235]	
[PRS_E2E_UC_00236]	
[PRS_E2E_UC_00238]	
[PRS_E2E_UC_00239]	
[PRS_E2E_UC_00308]	
[PRS_E2E_UC_00316]	

▽

△

Number	Heading
[PRS_E2E_UC_00317]	
[PRS_E2E_UC_00320]	
[PRS_E2E_UC_00321]	
[PRS_E2E_UC_00325]	
[PRS_E2E_UC_00351]	
[PRS_E2E_UC_00463]	
[PRS_E2E_UC_00464]	
[PRS_E2E_UC_00466]	
[PRS_E2E_UC_00606]	
[PRS_E2E_UC_01158]	
[PRS_E2E_UC_01168]	
[PRS_E2E_UC_01204]	
[PRS_E2E_USE_00741]	

Table B.22: Deleted Specification Items in R25-11

B.8 Traceable item history of this document according to AUTOSAR Release R26-11

B.8.1 Added Constraints in R26-11

none

B.8.2 Changed Constraints in R26-11

none

B.8.3 Deleted Constraints in R26-11

none

B.8.4 Added Specification Items in R26-11

none

B.8.5 Changed Specification Items in R26-11

none

B.8.6 Deleted Specification Items in R26-11

none