Document Title	Specification of CAN Interface
Document Category	Software specification (SWS)
Document UID	12
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• CANsec Support • Support of zero-copy operation • Editorial changes
2025-11-27	R25-11	AUTOSAR Release Management	• Abstraction from Driver APIs • Removal of TTCan • Improve mask based reception and transmission • Editorial changes
2024-11-27	R24-11	AUTOSAR Release Management	• Replaced upper communication layers by LSduR • Changed available via header file • Enhanced chapter Security Events • Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Support for selective WakeUp via CAN-Controller • Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• CAN XL Support • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Support hardware-based timestamping • Minor corrections/clarifications • Editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• Introduction of Security Events • Rework of Error classification chapters • Removal of pretended networking • Minor corrections/clarifications

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Update reference to ISO 11898-1:2015 • Minor corrections • Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• BusMirroring (CONC_634) • Receive Data Length Check per Pdu • Remove dummy implementations for Cancel Transmit APIs • Header File Cleanup
2017-12-08	4.3.1	AUTOSAR Release Management	• Introduction of Runtime errors • Replace Can_ReturnType with Std_ReturnType overlay • Minor corrections • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Remove CCMSM • Rework MetaData handling • Reliable TxConfirmation • Error Active/Passive State API
2015-07-31	4.2.2	AUTOSAR Release Management	• Clarified wakeup, buffering, transmit, and variants • Removed deprecated APIs • Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Full CAN FD Support • Global Time Synchronization over CAN • Removed CanIf_CancelTxConfirmation • Small improvements
2014-03-31	4.1.3	AUTOSAR Release Management	• Removed BSW Exclusive areas • Set ICOM support to optional • `Can_IdType` handling • Small improvements

▽

△

2013-10-31	4.1.2	AUTOSAR Release Management	• Restricted PDU mode changes • Removed critical section handling description in Chapter 9 • Set CanIfInitRefCfgSet oboslete • Pretended Networking section • Small improvements
2013-03-15	4.1.1	AUTOSAR Administration	• CAN FD (without DLC extension) • Pretended Networking (ICOM) • Heavy Duty Vehicle (J1939) support • PduModes and PnTxFilter for clean wake-up • Relation between PDUs & HOHs • Post-build loadable concept
2011-12-22	4.0.3	AUTOSAR Administration	• Partial Networking Support • Improved Transmit Buffering • Improved Error Detection
2009-12-18	4.0.1	AUTOSAR Administration	• Updated chapters "Version Checking" and "Published Information" • Multiple CAN IDs could optionally be assigned to one I-PDU • Wake-up validation optionally only via NM PDUs • Asynch. mode indication call-backs instead of synch. mode changes • No automatic PDU channel mode change when CC mode changes • TxConfirmation state entered for BusOff Recovery • WakeupSourceRefIn and WakeupSourceRefOut • PduInfoPtr instead of SduDataPtr • Introduction of Can_GeneralTypes.h and Can_HwHandleType • Transceiver types of chapter 8. shifted to transceiver SWS

▽

△

2010-02-02	3.1.4	AUTOSAR Administration	• HOH definition • abstracted ControllerId and TransceiverId • No changing of baudrate via CanIf and CanIf_ControllerInit • Dispatcher adapted because of CDD • TxBuffering: only one buffer per L-PDU • Wake up mechanism adapted to environment behavior (network -> controller/transceiver; wakeupSource) • Mode changes made asynchronous • no complete state machine in CanIf, just buffered states per controller • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2008-02-01	3.0.2	AUTOSAR Administration	• Replaced chapter 10 content with generated tables from AUTOSAR MetaModel.
2008-02-01	3.0.2	AUTOSAR Administration	• Interface abstraction: network related interface changed into a controller related one • Wakeup mechanism completely reworked, APIs added & changed for Wakeup • Initialization changed (flat initialization) • Scheduled main functions skipped due to changed BSW Scheduler responsibility • Document meta information extended • Small layout adaptations made

▽

△

2007-12-21	3.0.1	AUTOSAR Administration	• Header file structure changed • Support of mixed mode operation (Standard CAN & Extended CAN in parallel on one network) added • Support of CAN Transceiver API <User>_DlcErrorNotification deleted • Pre-compile/Link-Time/Post-Built definition for configuration parameters partly changed • Re-entrant interface call allowed for certain APIs • Support of AUTOSAR BSW Scheduler added • Support of memory mapping added • Configuration container structure reworked • Various of clarification extensions and corrections
2006-05-16	2.0.0	AUTOSAR Administration	• Second Release
2005-05-31	1.0.0	AUTOSAR Administration	• 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 15
3 Related documentation 18
3.1 Input documents & related standards and norms . . . 18
3.2 Related specification . . . 19
4 Constraints and assumptions 20
4.1 Limitations . . . 20
4.2 Applicability to car domains . . . 20
5 Dependencies to other modules 21
5.1 Upper Protocol Layers . . . 22
5.2 Initialization: Ecu State Manager . . . 22
5.3 Mode Control: CAN State Manager . . . 22
5.4 Lower layers: CAN Driver . . . 22
5.5 Lower layers: CAN Transceiver Driver . . . 23
5.6 Configuration . . . 24
5.7 File structure . . . 25
5.7.1 Code file structure . . . 25
5.7.2 Header file structure . . . 25
6 Requirements Tracing 26
7 Functional specification 27
7.1 General Functionality . . . 27
7.2 Hardware object handles . . . 28
7.3 Static L-PDUs . . . 30
7.4 Dynamic L-PDUs . . . 31
7.4.1 Dynamic Transmit L-PDUs . . . 32
7.4.2 Dynamic receive L-PDUs . . . 33
7.5 Physical channel view . . . 33
7.6 CAN Hardware Unit . . . 35
7.7 BasicCAN and FullCAN reception . . . 36
7.8 Initialization . . . 37
7.9 Transmit request . . . 38
7.10 Transmit data flow . . . 39
7.11 Transmit buffering . . . 40
7.11.1 General behavior . . . 40
7.11.2 Buffer characteristics . . . 41
7.11.2.1 Storage of L-PDUs in the transmit L-PDU buffer . . . 41
7.11.2.2 Clearance of transmit L-PDU buffers . . . 43
7.11.2.3 Initialization of transmit L-PDU buffers . . . 43

7.11.3 Data integrity of transmit L-PDU buffers . . . 43
7.12 Transmit confirmation . . . 44
7.13 Receive data flow . . . 44
7.14 Receive indication . . . 46
7.15 Read received data . . . 48
7.16 Read Tx/Rx notification status . . . 49
7.17 Data integrity . . . 49
7.18 CAN Controller Mode . . . 50
7.18.1 General Functionality . . . 50
7.18.2 CAN Controller Operation Modes . . . 51
7.18.3 Controller Mode Transitions . . . 52
7.18.4 Wake-up . . . 53
7.18.4.1 Wake-up detection . . . 53
7.18.4.2 Wake-up Validation . . . 53
7.19 PDU channel mode control . . . 54
7.19.1 PDU channel groups . . . 54
7.19.2 PDU channel modes . . . 55
7.19.2.1 CANIF_OFFLINE . . . 56
7.19.2.2 CANIF_ONLINE . . . 57
7.19.2.3 CANIF_OFFLINE_ACTIVE . . . 58
7.20 Software receive filter . . . 58
7.20.1 Software filtering concept . . . 59
7.20.2 Software filter algorithms . . . 61
7.21 Data Length Check . . . 61
7.22 Polling mode . . . 62
7.23 Multiple CAN Driver support . . . 62
7.23.1 Transmit requests by using multiple CAN Drivers . . . 62
7.23.2 Notification mechanism using multiple CAN Drivers . . . 64
7.24 Partial Networking . . . 66
7.25 CAN FD Support . . . 67
7.26 CAN XL Support . . . 68
7.26.1 Dynamic Control of CAN XL Attributes . . . 69
7.27 CANsec Support . . . 69
7.27.1 Communication via controlled and uncontrolled port . . . 71
7.28 Security Events . . . 72
7.29 Error classification . . . 74
7.29.1 Development Errors . . . 74
7.29.2 Runtime Errors . . . 74
7.29.3 Production Errors . . . 75

8 API specification 76
8.1 Imported types . . . 76
8.2 Type definitions . . . 77
8.2.1 CanIf_ConfigType . . . 77

8.2.2 CanIf_PduModeType . . . 77
8.2.3 CanIf_NotifStatusType . . . 78
8.3 Function definitions . . . 78
8.3.1 CanIf_Init . . . 78
8.3.2 CanIf_DeInit . . . 79
8.3.3 CanIf_SetControllerMode . . . 79
8.3.4 CanIf_GetControllerMode . . . 80
8.3.5 CanIf_GetControllerErrorState . . . 81
8.3.6 CanIf_Transmit . . . 82
8.3.7 CanIf_ReleaseRxBuffer . . . 85
8.3.8 CanIf_ReadRxPduData . . . 86
8.3.9 CanIf_ReadTxNotifStatus . . . 87
8.3.10 CanIf_ReadRxNotifStatus . . . 88
8.3.11 CanIf_SetPduMode . . . 89
8.3.12 CanIf_GetPduMode . . . 90
8.3.13 CanIf_SetDynamicTxId . . . 91
8.3.14 CanIf_SetTrcvMode . . . 92
8.3.15 CanIf_GetTrcvMode . . . 93
8.3.16 CanIf_GetTrcvWakeupReason . . . 95
8.3.17 CanIf_SetTrcvWakeupMode . . . 96
8.3.18 CanIf_CheckWakeup . . . 98
8.3.19 CanIf_CheckValidation . . . 99
8.3.20 CanIf_GetTxConfirmationState . . . 100
8.3.21 CanIf_ClearTrcvWufFlag . . . 101
8.3.22 CanIf_CheckTrcvWakeFlag . . . 101
8.3.23 CanIf_SetBaudrate . . . 102
8.3.24 CanIf_GetControllerRxErrorCounter . . . 103
8.3.25 CanIf_GetControllerTxErrorCounter . . . 104
8.3.26 CanIf_EnableBusMirroring . . . 105
8.3.27 CanIf_GetCurrentTime . . . 105
8.3.28 CanIf_EnableEgressTimeStamp . . . 107
8.3.29 CanIf_GetEgressTimeStamp . . . 108
8.3.30 CanIf_GetIngressTimeStamp . . . 109
8.3.31 CanIf_TriggerTransmit . . . 110
8.3.32 CanIf_TxConfirmation . . . 111
8.3.33 CanIf_RxIndication . . . 112
8.3.34 CanIf_XLRxIndication . . . 113
8.3.35 CanIf_XLReportErrorState . . . 114
8.3.36 CanIf_ControllerBusOff . . . 115
8.3.37 CanIf_ConfirmPnAvailability . . . 116
8.3.38 CanIf_ConfirmCtrlPnAvailability . . . 117
8.3.39 CanIf_ClearTrcvWufFlagIndication . . . 118
8.3.40 CanIf_CheckTrcvWakeFlagIndication . . . 119

8.3.41 CanIf_ControllerModeIndication . . . 120
8.3.42 CanIf_TrcvModeIndication . . . 121
8.3.43 CanIf_ControllerErrorStatePassive . . . 122
8.3.44 CanIf_ErrorNotification . . . 123
8.3.45 CanIf_CanSecUpdateSecY . . . 124
8.3.46 CanIf_CanSecAddRxSc . . . 124
8.3.47 CanIf_CanSecDeleteRxSc . . . 125
8.3.48 CanIf_CanSecAddTxSa . . . 125
8.3.49 CanIf_CanSecUpdateTxSa . . . 126
8.3.50 CanIf_CanSecDeleteTxSa . . . 127
8.3.51 CanIf_CanSecAddRxSa . . . 128
8.3.52 CanIf_CanSecUpdateRxSa . . . 129
8.3.53 CanIf_CanSecDeleteRxSa . . . 129
8.3.54 CanIf_CanSecGetTxSaNextPn . . . 130
8.3.55 CanIf_CanSecGetCanSecSecYStatistics . . . 131
8.3.56 CanIf_CanSecGetCanSecTxScStatistics . . . 131
8.3.57 CanIf_CanSecGetCanSecRxScStatistics . . . 132
8.3.58 CanIf_CanSecSetControlledPortEnabled . . . 133
8.3.59 CanIf_CanSecUpdateSecYNotification . . . 133
8.3.60 CanIf_CanSecAddTxSaNotification . . . 134
8.3.61 CanIf_CanSecAddRxSaNotification . . . 135
8.3.62 CanIf_CanSecGetCanSecSecYStatisticsNotification . . . 135
8.3.63 CanIf_CanSecGetCanSecTxScStatisticsNotification . . . 136
8.3.64 CanIf_CanSecGetCanSecRxScStatisticsNotification . . . 137
8.4 Scheduled functions . . . 137
8.5 Expected interfaces . . . 137
8.5.1 Mandatory interfaces . . . 137
8.5.2 Optional interfaces . . . 138
8.5.3 Configurable interfaces . . . 141
8.5.3.1 <User_ValidateWakeupEvent> . . . 142
8.5.3.2 <User_ControllerBusOff> . . . 143
8.5.3.3 <User_ConfirmPnAvailability> . . . 145
8.5.3.4 <User_ConfirmCtrlPnAvailability> . . . 146
8.5.3.5 <User_ClearTrcvWufFlagIndication> . . . 147
8.5.3.6 <User_CheckTrcvWakeFlagIndication> . . . 148
8.5.3.7 <User_ControllerModeIndication> . . . 149
8.5.3.8 <User_TrcvModeIndication> . . . 151
9 Sequence diagrams 153
9.1 Transmit request (single CAN Driver) . . . 153
9.2 Transmit request (multiple CAN Drivers) . . . 154
9.3 Transmit confirmation (interrupt mode) . . . 156
9.4 Transmit confirmation (polling mode) . . . 157
9.5 Transmit confirmation (with buffering) . . . 158

9.6 Trigger Transmit Request . . . 159
9.7 Receive indication (interrupt mode) . . . 161
9.8 Receive indication (polling mode) . . . 163
9.9 Read received data . . . 165
9.10 Start CAN network . . . 167
9.11 BusOff notification . . . 169
9.12 BusOff recovery . . . 170

10 Configuration specification 172
10.1 Containers and configuration parameters . . . 172
10.1.1 CanIf . . . 173
10.1.2 CanIfPrivateCfg . . . 174
10.1.3 CanIfPublicCfg . . . 176
10.1.4 CanIfInitCfg . . . 187
10.1.5 CanIfTxPduCfg . . . 190
10.1.6 CanIfTxPduXLParams . . . 195
10.1.7 CanIfRxPduCfg . . . 197
10.1.8 CanIfRxPduXLParams . . . 201
10.1.9 CanIfRxPduCanIdRange . . . 203
10.1.10 CanIfDispatchCfg . . . 205
10.1.11 CanIfCtrlCfg . . . 213
10.1.12 CanIfCtrlDrvCfg . . . 215
10.1.13 CanIfTrcvDrvCfg . . . 216
10.1.14 CanIfTrcvCfg . . . 217
10.1.15 CanIfInitHohCfg . . . 219
10.1.16 CanIfHthCfg . . . 220
10.1.17 CanIfHrhCfg . . . 221
10.1.18 CanIfHrhRangeCfg . . . 223
10.1.19 CanIfBufferCfg . . . 226
10.1.20 CanIfSecurityEventRefs . . . 228
10.2 Configuration constraints . . . 230

A Not applicable requirements 233

B Change History 234
B.1 Change History of this document according to AUTOSAR Release R23-11 234
B.1.1 Added Specification Items in R23-11 . . . 234
B.1.2 Changed Specification Items in R23-11 . . . 234
B.1.3 Deleted Specification Items in R23-11 . . . 235
B.1.4 Added Constraints in R23-11 . . . 235
B.1.5 Changed Constraints in R23-11 . . . 235
B.1.6 Deleted Constraints in R23-11 . . . 236
B.2 Change History of this document according to AUTOSAR Release R24-11 237
B.2.1 Added Specification Items in R24-11 . . . 237
B.2.2 Changed Specification Items in R24-11 . . . 237

B.2.3 Deleted Specification Items in R24-11 . . . 238
B.2.4 Added Constraints in R24-11 . . . 239
B.2.5 Changed Constraints in R24-11 . . . 239
B.2.6 Deleted Constraints in R24-11 . . . 239
B.3 Change History of this document according to AUTOSAR Release R25-11 240
B.3.1 Added Specification Items in R25-11 . . . 240
B.3.2 Changed Specification Items in R25-11 . . . 240
B.3.3 Deleted Specification Items in R25-11 . . . 244
B.3.4 Added Constraints in R25-11 . . . 245
B.3.5 Changed Constraints in R25-11 . . . 245
B.3.6 Deleted Constraints in R25-11 . . . 245
B.4 Change History of this document according to AUTOSAR Release R26-11 246
B.4.1 Added Specification Items in R26-11 . . . 246
B.4.2 Changed Specification Items in R26-11 . . . 247
B.4.3 Deleted Specification Items in R26-11 . . . 252
B.4.4 Added Constraints in R26-11 . . . 253
B.4.5 Changed Constraints in R26-11 . . . 253
B.4.6 Deleted Constraints in R26-11 . . . 254

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module CAN Interface.

As depicted in Figure 1.1 the CAN Interface module is located between the low level CAN device drivers (CAN Driver [1] and Transceiver Driver [2]) and the upper communication service layers (i.e. CAN State Manager [3], CAN Network Management [4], CAN Transport Protocol [5], PDU Router [6]). It represents the interface to the services of the CAN Driver for the upper communication layers.

The CAN Interface module provides a unique interface to manage different CAN hardware device types like CAN Controllers and CAN Transceivers used by the defined ECU hardware layout. Thus multiple underlying internal and external CAN Controllers/CAN Transceivers can be controlled by the CAN State Managers module based on a physical CAN channel related view.

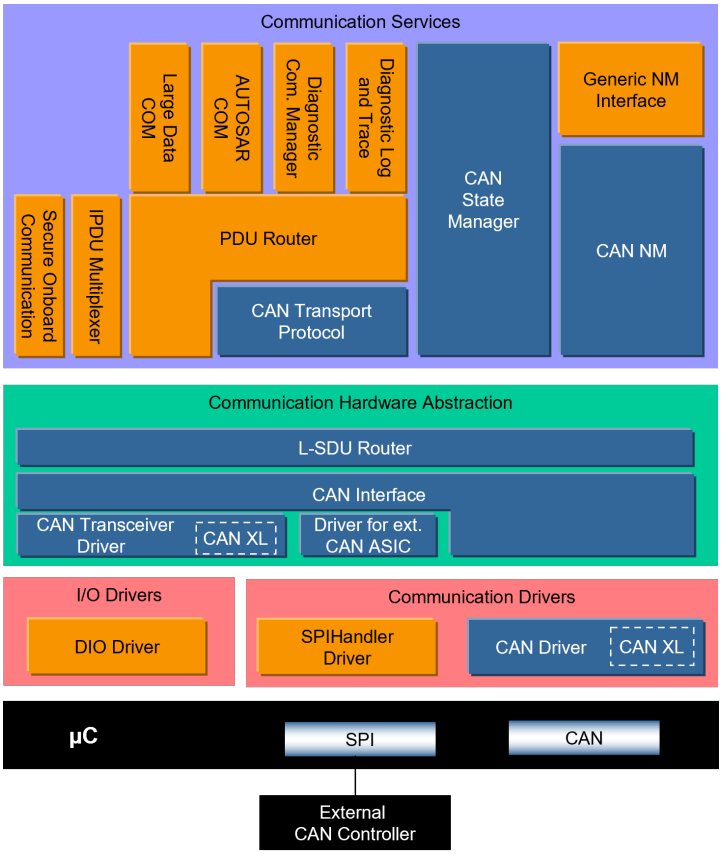


Figure 1.1: AUTOSAR CAN Layered Architecture

The CAN Interface module consists of all CAN hardware independent tasks, which belongs to the CAN communication device drivers of the corresponding ECU. Those functionality is implemented once in the CAN Interface module, so that underlying CAN device drivers only focus on access and control of the corresponding specific CAN hardware device.

CanIf fulfils main control flow and data flow requirements of the PDU Router and upper layer communication modules of the AUTOSAR COM stack: *transmit request processing*, *transmit confirmation* / *receive indication* / *error notification* and *start* / *stop* of a CAN Controller and thus *waking up* / *participating on a network*. Its data processing and notification API is based on CAN L-SDUs, whereas APIs for control and mode handling provides a CAN Controller related view.

In case of Transmit Requests CanIf completes the L-PDU transmission with corresponding parameters and relays the CAN L-PDU via the appropriate CanDrv to the CAN Controller. At reception CanIf distributes the Received L-PDUs as L-SDUs to the upper layer. The assignment between Receive L-SDU and upper layer is statically configured. At transmit confirmation CanIf is responsible for the notification of upper layers about successful transmission.

The CAN Interface module provides CAN communication abstracted access to the CAN Driver and CAN Transceiver Driver services for control and supervision of the CAN network. The CAN Interface forwards downwards the status change requests from the CAN State Manager to the lower layer CAN device drivers, and upwards the CAN Driver / CAN Transceiver Driver events are forwarded by the CAN Interface module to e.g. the corresponding NM module.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the CAN Interface module that are not included in the [7, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
CAN L-PDU	CAN Protocol Data Unit. Consists of an identifier, Data Length and data (SDU) Visible to the CAN driver.
CAN L-SDU	CAN Service Data Unit. Data that are transported inside the CAN L-PDU. Visible to the upper layers of the CAN interface (e.g. PDU Router).
CanDrv	CAN Driver module
CAN FD	CAN with Flexible Data-Rate
CanId	CAN Identifier
CanIf	CAN Interface module
CanNm	CAN Network Management module
CanSm	CAN State Manager module
CanTp	CAN Transport Layer module
CanTrcv	CAN Transceiver Driver module
CanTSyn	Global Time Synchronization over CAN
ComM	Communication Manager module
DCM	Diagnostic Communication Manager module
EcuM	ECU State Manager module
HOH	CAN hardware object handle
HRH	CAN hardware receive handle
HTH	CAN hardware transmit handle
J1939Nm	J1939 Network Management module
J1939Tp	J1939 Transport Layer module
LLC	logical link control
L-SDU Router	Module that transfers L-SDUs from one module to another module. The L-SDU Router module can be utilized for internal routing purposes.
PduR	PDU Router module
PN	Partial Networking
SchM	Scheduler Module
SDT	SDU type
SDU	service data unit
VCID	Virtual CAN network ID
XLFF	XL Frame Format

Abbreviation / Acronym:	Description:
Buffer	Fixed sized memory area for a single data unit (e.g. CAN ID, Data Length, SDU, etc.) is stored at a dedicated memory address in RAM.
CAN communication matrix	Describes the complete CAN network: • Participating nodes • Definition of all CAN PDUs (Identifier, Data Length) • Source and Sinks for PDUs
CAN Controller	A CAN Controller is a CPU on-chip or external standalone hardware device. One CAN Controller is connected to one physical channel.

CAN Device Driver	Generic term of CAN Driver and CAN Transceiver Driver.
CAN Hardware Unit	A CAN Hardware Unit may consist of one or multiple CAN Controllers of the same type and one, two or multiple CAN RAM areas. The CAN Hardware Unit is located on-chip or as external device. The CAN hardware unit is represented by one CAN Driver.
CanIf Controller mode state machine	This is not really a state machine, which may be influenced by transmission requests. This is an image of the current abstracted state of an appropriate CAN Controller. The state transitions can only be realized by upper layer modules like the CanSm or by external events like e.g. if a BusOff occurred.
CanIf Receive L-PDU / CanIf Rx L-PDU	L-PDU of which the direction is set to "lower to upper layer".
CanIf Receive L-PDU buffer / CanIfRxBuffer	Single element RAM buffer located in the CAN Interface module to store whole receive L-PDUs.
CanIf Transmit L-PDU / CanIf Tx L-PDU	L-PDU of which the direction is set to "upper to lower layer".
CanIf Transmit L-PDU buffer / CanIfTxBuffer	Single CanIfTxBuffer element located in the CanIf to store one or multiple CanIf Tx L-PDUs. If the buffersize of a single CanIfTxBuffer element is set to 0, a CanIfTxBuffer element is only used to refer a HTH.
ControllerId	Identifier provided by the CAN Interface module which abstracts from the different Controllers of the different CAN Driver instances.
Hardware object / HW object	A CAN hardware object is defined as a PDU buffer inside the CAN RAM of the CAN Hardware Unit / CAN Controller.
Hardware Receive Handle (HRH)	The Hardware Receive Handle (HRH) is defined and provided by the CAN Driver. Each HRH typically represents just one hardware object. The HRH is used as a parameter by the CAN Interface Layer for i.e. software filtering.
Hardware Transmit Handle (HTH)	The Hardware Transmit Handle (HTH) is defined and provided by the CAN Driver. Each HTH typically represents just one or multiple CAN hardware objects that are configured as CAN hardware transmit buffer pool.
Inner priority inversion	Transmission of a high-priority L-PDU is prevented by the presence of a pending low-priority L-PDU in the same transmit hardware object.
Integration Code	Code that the Integrator needs to add to an AUTOSAR System, to adapt non-standardized functionalities. Examples are Callouts of the ECU State Manager and Callbacks of various other BSW modules. The I/O Hardware Abstraction is called Integration Code, too.
Lowest In - First Out / LOFO	This is a data storage procedure, whereas always the elements with the lowest values will be extracted.
L-PDU channel group	Group of CAN L-PDUs, which belong to just one underlying network. Usually they are handled by one upper layer module.
Outer priority inversion	A time gap occurs between two consecutive transmit L-PDUs. In this case a lower priority L-PDU from another node can prevent sending the own higher priority L-PDU. Here the higher priority L-PDU cannot participate in arbitration during network access because the lower priority L-PDU already won the arbitration.
Physical channel	A physical channel represents an interface from a CAN Controller to the CAN Network. Different physical channels of the CAN Hardware Unit may access different networks.

PnTxFilter	Transmit filter per CAN Controller which overlays the PDU channel modes.
TransceiverId	Identifier provided by the CAN Interface module which abstracts from the different Transceivers of the different CAN Transceiver Driver instances.
Tx request	Transmit request to the CAN Interface module from a upper layer module of the CanIf

3 Related documentation

3.1 Input documents & related standards and norms

[1] Specification of CAN Driver
AUTOSAR_CP_SWS_CANDriver

[2] Specification of CAN Transceiver Driver
AUTOSAR_CP_SWS_CANTransceiverDriver

[3] Specification of CAN State Manager
AUTOSAR_CP_SWS_CANStateManager

[4] Specification of CAN Network Management
AUTOSAR_CP_SWS_CANNetworkManagement

[5] Specification of CAN Transport Layer
AUTOSAR_CP_SWS_CANTransportLayer

[6] Specification of PDU Router
AUTOSAR_CP_SWS_PDURouter

[7] Glossary
AUTOSAR_FO_TR_Glossary

[8] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[9] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[10] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[11] ISO 11898-1:2015 – Road vehicles – Controller area network (CAN)

[12] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[13] CiA 610-1 version 1.0.0 (DSP) - CAN XL specifications and test plans - Part 1: Data link layer and physical coding sub-layer requirements
http://www.can-cia.org

[14] CiA 611-1 version 1.0.0 (DSP) - CAN XL higher layer functions - Part 1: Definition of service data unit types
http://www.can-cia.org

[15] CiA 613-2 version 1.0.0 (DSP) - CAN XL add-on services - Part 2: Security
http://www.can-cia.org

[16] IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security

https://ieeexplore.ieee.org/document/8585421

[17] IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control
https://ieeexplore.ieee.org/document/9018454

[18] Specification of MACsec Key Agreement
AUTOSAR_CP_SWS_MACsecKeyAgreement

[19] Specification of ECU Configuration
AUTOSAR_CP_TPS_ECUConfiguration

[20] System Template
AUTOSAR_CP_TPS_SystemTemplate

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [8, SWS BSW General], which is also valid for CAN Interface.

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

4 Constraints and assumptions

4.1 Limitations

The CAN Interface can be used for CAN communication only and is specifically designed to operate with one or multiple underlying CAN Drivers and CAN Transceiver Drivers. Several CAN Driver modules covering different CAN Hardware Units are represented by just one generic interface as specified in the CAN Driver specification [1]. As well in the same manner several CAN Transceiver Driver modules covering different CAN Transceiver devices are represented by just one generic interface as specified in the CAN Transceiver Driver specification [2, Specification of CAN Transceiver Driver]. Other protocols than CAN (i.e. LIN or FlexRay) are not supported.

Please be aware that an active PnTxFilter ensures that the first messages on bus is CanIfTxPduPnFilterPdu. In case that CanIfTxPduPnFilterPdu is the NM-PDU the COM-Stack start up takes care that the PduGroups are disabled until successful transmission of that PDU. However, transmit requests for other PDUs (i.e. initially started PDUs, TP-PDUs, XCP-PDUs) will be rejected until the configured PDU was sent. Only the very first PDU which initiates the Wake-up of the Network has to be the CanIfTxPduPnFilterPdu. In case communication is ongoing and there is an successful reception of frame with PnTxFilter enabled, PnTxFilter shall be disabled. The PnTxFilter is in this case not needed since an Ack will be provided by an already active Node.

Note that AUTOSAR does not specify a `CANsec` KaY. An extension of the MACsec Key Agreement (MKA) module to cover also CANsec key agreement is planned for future releases, once the `CANsec` key agreement is specified by CiA. The respective APIs are already specified in the MKA module, but they are non-functional and need to be implemented by a CDD.

4.2 Applicability to car domains

The CAN Interface can be used for all domain applications when the CAN protocol is used.

5 Dependencies to other modules

This section describes the relations to other modules within the AUTOSAR basic software architecture. It contains brief descriptions of configuration information and services, which are required by the CAN Interface Layer from other modules (see Figure 5.1).

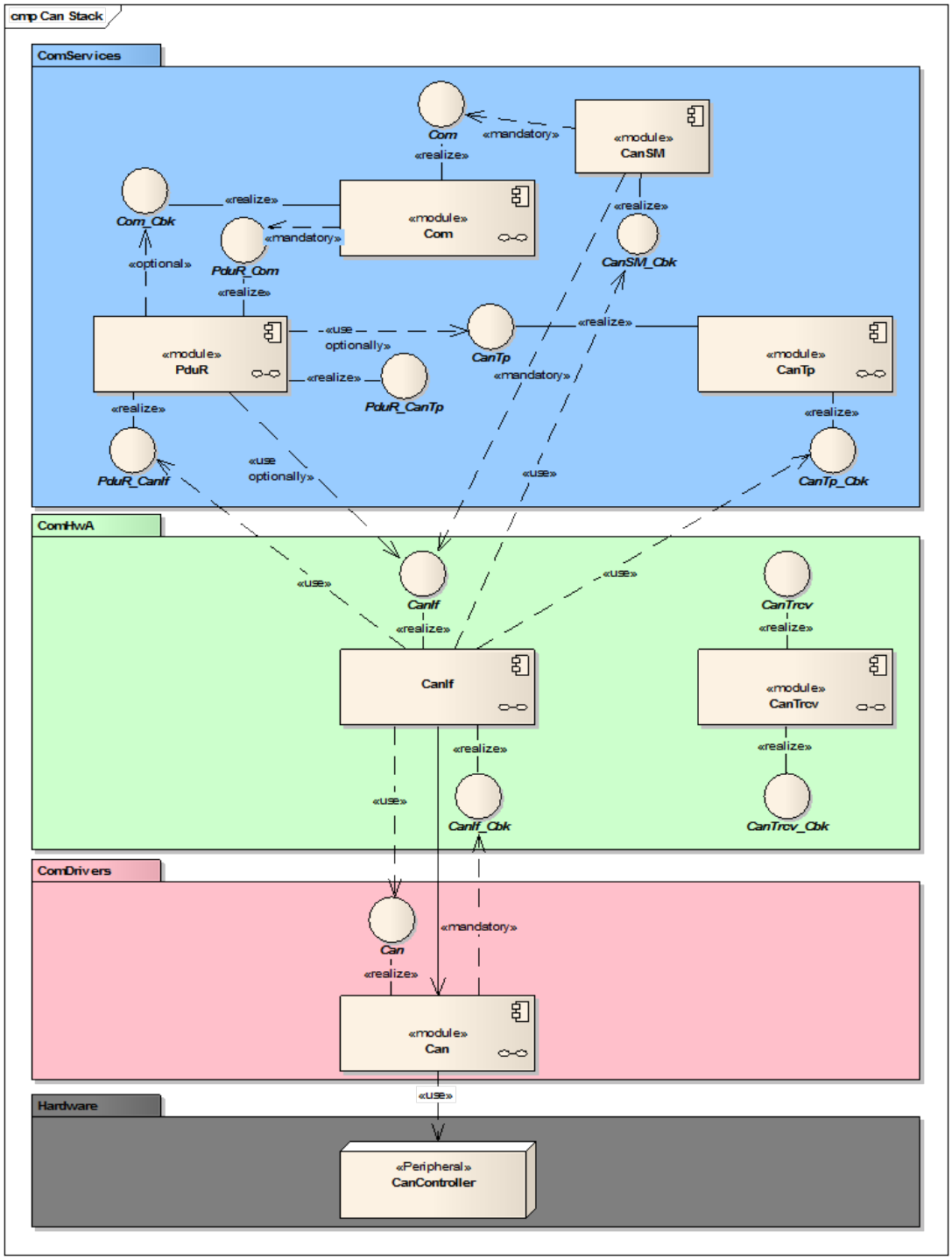


Figure 5.1: CANIF dependencies in AUTOSAR BSW

5.1 Upper Protocol Layers

Inside the AUTOSAR BSW architecture the upper layers of the CAN Interface module (Abbr.: CanIf) are represented by the PDU Router module (Abbr.: PduR), CAN Network Management module (Abbr.: CanNm), CAN Transport Layer module (Abbr.: CanTp), CAN State Manager module (Abbr.: CanSm), ECU State Manager module (Abbr.: EcuM), Complex Driver modules (Abbr.: *CDD*), Universal Calibration Protocol module (Abbr.: *XCP*), Global Time Synchronization over CAN (Abbr.: CanTSyn), J1939 Transport Layer module (Abbr.: J1939Tp) and J1939 Network Management module (Abbr.: J1939Nm).

The AUTOSAR BSW architecture indicates that the application data buffers are located in the upper layer, to which they belong. Direct access to these buffers is prohibited. The buffer location is passed by the CanIf from or to the CAN Driver module (Abbr.: CanDrv) during transmission and reception. During execution of these transmission/reception indication services buffer location is passed. Data integrity is guaranteed by use of lock mechanisms each time the buffer has been accessed. See Chapter 7.17 “Data integrity”.

The API used by the CanIf consists of notification services as basic agents for the transfer of CAN related data (i.e. Data Length) to the target upper layer. The call parameters of these services points to the information buffered in the CanDrv or they refer directly to the CAN Hardware.

In addition, the CanIf supports a callout to the Bus Mirroring module, to report the content of received and transmitted frames.

5.2 Initialization: Ecu State Manager

The EcuM initializes the CanIf (refer to [3, Specification of ECU State Manager]).

5.3 Mode Control: CAN State Manager

The CanSm module is responsible for mode control management of all supported CAN Controllers and CAN Transceivers.

5.4 Lower layers: CAN Driver

The main lower layer CAN device driver is represented by the CanDrv (see [1, Specification of CAN Driver]). The CanIf has a close relation to the CanDrv as a result of its position in the AUTOSAR Basic Software Architecture.

The CanDrv provides a hardware abstracted access to the CAN Controller only, but control of operation modes is done in CanSm only.

The CanDrv detects and processes events of the CAN Controllers and notifies those to the CanIf.

The CanIf passes operation mode requests of the CanSm to the corresponding underlying CAN Controllers.

CanDrv provides a normalized L-PDU to ensure hardware independence of CanIf. The pointer to this normalized L-PDU points either to a temporary buffer (for e.g. data normalizing) or to the CAN hardware dependent CanDrv. For CanIf the kind of L-PDU buffer is invisible.

The CanIf provides notification services used by the CanDrv in all notifications scenarios, for example: *transmit confirmation* (CanIf_TxConfirmation()), *receive indication* (CAN 2.0/FD CanIf_RxIndication(), CAN XL CanIf_XLRxIndication()) and *notification of a controller mode change* CanIf_ControllerModeIndication().

In case of using multiple CanDrv serving different interrupt vectors these callback services mentioned above must be re-entrant, refer to Chapter 7.23 “Multiple CAN Driver support”. Reentrancy of callback functions is specified in Chapter 8.3.

The callback services called by the CanDrv are declared and implemented inside the CanIf. The callback services called by the CanIf are declared and placed inside the appropriate upper communication service layer, for example PduR, CanNm, CanTp. The CanIf structure is specified in Chapter 5.7 “File structure”.

The number of configured CAN Controllers does not necessarily belong to the number of used CAN Transceivers. In case multiple CAN Controllers of a different types operate on the same CAN network, one CAN Transceiver and CanTrcv is sufficient, whereas dependent to the type of the CAN Controller devices one or two different CanDrv are needed (see Chapter 7.5 “Physical channel view”).

5.5 Lower layers: CAN Transceiver Driver

The second available lower layer CAN device driver is represented by the CanTrcv (see [2, Specification of CAN Transceiver Driver]).

Each CanTrcv itself does operation mode control of the CAN Transceiver device. The CanIf just maps all APIs of several underlying CanTrcvs to a unique one, thus CanSm is able to trigger a transition of the corresponding CAN Transceiver modes. No control or handling functionality belonging to CanTrcv is done inside the CanIf.

The CanIf maps the following services of all underlying CanTrcvs to one unique interface. These are further described in the CAN Transceiver Driver SWS (see [2, Specification of CAN Transceiver Driver]):

- Unique CanTrcv mode request and read services to manage the operation modes of each underlying CAN Transceiver device.

- Read service for CAN Transceiver *wake up reason* support.
- Mode request service to *enable*/*disable*/*clear* wake up event state of each used CAN transceiver (`CanIf_SetTrcvMode()`, see [SWS_CANIF_00287]).

5.6 Configuration

The `CanIf` design is optimized to manage CAN protocol specific capabilities and handling of the used underlying CAN Controller.

The CanIf is capable to change the CAN configuration without a *re-build*. Therefore, the function `CanIf_Init()` (see [SWS_CANIF_00001]) retrieves the required CAN configuration information from configuration containers and parameters, which are specified (linked as references, or additional parameters) in Chapter 10, see Figure 10.1.

This section gives a summary of the retrieved information, e.g.:

- Number of CAN Controllers. The number of CAN Controllers is necessary for dispatching of transmit and receive L-PDUs and for the control of the status of the available CAN Drivers (see `CanIfCtrlDrvCfg`).
- Number of Hardware Object Handles. To supervise transmit requests the CAN Interface needs to know the number of HTHs and the assignments between each HTH and the corresponding CAN Controller (see `CanIfHthCanCtrlIdRef`; `CanIfHthIdSymRef`).
- Range of received CAN IDs passing hardware acceptance filter for each hardware object. The CAN Interface uses fixed assignments between HRHs and L-PDUs to be received in the corresponding hardware object to conduct a search algorithm (see Chapter 7.20 “Software receive filter”, see `CanIfHrhSoftwareFilter`, `CanIfHrhCanCtrlIdRef`, `CanIfHrhIdSymRef`)

`CanIf` needs information about all used upper communication service layers and `L-SDUs` to be dispatched. The following information has to be set up at configuration time for integration of `CanIf` inside the AUTOSAR COM stack:

- Transmitting upper layer module and transmit *I-PDU* for each transmit `L-SDU`.
  => Used for dispatching of transmit confirmation services
  (see `CanIfTxPduId`).
- Receiving upper layer module and receive *I-PDU* for each receive `L-SDU`.
  => Used for `L-SDU` dispatching during receive indication
  (see `CanIfRxPduId`).

The CanIf needs the description of the controller and the own ECU, which is connected to one or multiple CAN networks. The following information is therefore retrieved from the CAN communication matrix, part of the AUTOSAR system configuration (see `CanIfTxPduCfg`, `CanIfRxPduCfg`):

- All L-PDUs received on each physical channel of this ECU.
  => Used for software filtering and receive L-SDU dispatch
- All L-SDUs that shall be transmitted by each physical channel on this ECU.
  => Used for the transmit request and Transmit L-PDU dispatch
- Properties of these L-PDUs (ID, Data Length).
  => Used for software filtering, receive indication services, Data Length Check
- Transmitter for each transmitted L-SDU (i.e. PduR, CanNm, CanTp).
  => Used for the transmit confirmation services
- Receiver for each receive L-SDU (i.e. PduR, CanNm, CanTp)
  => Used for the L-PDU dispatch
- Symbolic L-PDU/L-SDU name.
  => Used for the representation of Rx/Tx data buffer addresses

5.7 File structure

5.7.1 Code file structure

[SWS_CANIF_00378] ⌈CanIf shall access the location of the API of all used underlying CanDrvs for link time configuration by a set of function pointers for each CanDrv.⌋

The values for the function pointers for each CanDrv are given at link time.

5.7.2 Header file structure

[SWS_CANIF_00672] ⌈The header file CanIf.h only contains extern declarations of constants, global data and services that are specified in CanIf.⌋

Constants, global data types and functions that are only used by CanIf internally, are declared within CanIf.c.

[SWS_CANIF_00903] ⌈CanIf shall include the header file Mirror.h if Bus Mirroring is enabled (see CanIfBusMirroringSupport).⌋

Note that in order to support CANsec, additional types are imported from Mka.h.

6 Requirements Tracing

The following tables references the requirements specified in [9] and links to the fulfillment of these. Please note that if column 'Satisfied by' is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_CANIF_00001]
[SRS_BSW_00310]	API naming convention	[SWS_CANIF_91018]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_CANIF_00064]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_CANIF_00311] [SWS_CANIF_00313] [SWS_CANIF_00319] [SWS_CANIF_00320] [SWS_CANIF_00325] [SWS_CANIF_00326] [SWS_CANIF_00331] [SWS_CANIF_00336] [SWS_CANIF_00341] [SWS_CANIF_00346] [SWS_CANIF_00352] [SWS_CANIF_00353] [SWS_CANIF_00364] [SWS_CANIF_00398] [SWS_CANIF_00404] [SWS_CANIF_00410] [SWS_CANIF_00416] [SWS_CANIF_00417] [SWS_CANIF_00419] [SWS_CANIF_00429] [SWS_CANIF_00535] [SWS_CANIF_00536] [SWS_CANIF_00537] [SWS_CANIF_00538] [SWS_CANIF_00648] [SWS_CANIF_00649] [SWS_CANIF_00650] [SWS_CANIF_00656] [SWS_CANIF_00657] [SWS_CANIF_00774] [SWS_CANIF_00860] [SWS_CANIF_00869] [SWS_CANIF_00898] [SWS_CANIF_00899] [SWS_CANIF_00907] [SWS_CANIF_00908] [SWS_CANIF_00909] [SWS_CANIF_00910] [SWS_CANIF_00912]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_CANIF_91002]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_CANIF_00142]
[SRS_BSW_00353]	All type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_CANIF_00142]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_CANIF_91018]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_CANIF_00001]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_CANIF_91018]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_CANIF_00001]
[SRS_BSW_00406]	API handling in uninitialized state	[SWS_CANIF_91018]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_CANIF_00001]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 General Functionality

The services of CanIf can be divided into the following main groups:

- Initialization
- Transmit request services
- Transmit confirmation services
- Reception indication services
- Controller mode control services
- PDU mode control services

Possible applications of CanIf:

i. Interrupt Mode
CanDrv processes interrupts triggered by the CAN Controller. CanIf, which is event based, is notified when an event occurs. In this case the relevant CanIf services are called within the corresponding *ISRs* in CanDrv.

ii. Polling Mode
CanDrv is triggered by the SchM and performs subsequent processes (*Polling Mode*). In this case Can_MainFunction_<Write/Read/BusOff/Wakeup/-Transceiver>() must be called periodically within a defined time interval. CanIf is notified by CanDrv about events (*Reception*, *Transmission*, *BusOff*, *Timeout*), that occurred in one of the CAN Controllers, equally to the interrupt driven operation. CanDrv is responsible for the update of the corresponding information which belongs to the occurred event in the CAN Controller, for example reception of a L-PDU.

iii. Mixed Mode: interrupt and polling driven CanDrv
The functionality can be divided between *interrupt driven* and *polling driven* operation mode depending on the used CAN Controllers.
Examples: Polling driven *FullCAN* reception and interrupt driven *BasicCAN* reception, polling driven transmit and interrupt driven reception, etc.

This specification describes a unique interface, which is valid for all three types of operation modes. Summarized, CanIf works in the same way, either if any events are processed on interrupt, task level or mixed. The only difference is the call context and probably the way of interruption of the notifications: *pre-emptive* or *co-operative*. All services are performed in accordance with the configuration.

The following paragraphs describe the functionality of CanIf.

7.2 Hardware object handles

Hardware Object Handles (HOH) for transmission (HTH) as well as for reception (HRH) represent an abstract reference to a *CAN mailbox structure*, that contains CAN related parameters such as CanId, DLC and data. Based on the CAN hardware buffer abstraction each Hardware Object is referenced in CanIf independent of the CAN hardware buffer layout. The HOH is used as a parameter in the calls of CanDrv's interface services and is provided by CanDrv's configuration and used by CanDrv as identifier for communication buffers of the CAN mailbox.

CanIf acts only as user of the Hardware Object Handle, but does not interpret it on the basis of hardware specific information. CanIf therefore remains independent of hardware.

[SWS_CANIF_00023] ⌈CanIf shall avoid direct access to hardware specific communication buffers and shall access it exclusively via CanDrv interface services.⌋

Rationale for [SWS_CANIF_00023]: CanIf remains independent of hardware, because CanDrv interfaces are called with HOH parameters, which abstract from the concrete CAN hardware buffer properties.

Each CAN Controller can provide multiple CAN Transmit Hardware Objects in the CAN mailbox. These can be logically linked to one entire pool of Hardware Objects (multiplexed Hardware Objects) and thus addressed by one HTH.

[SWS_CANIF_00662] ⌈CanIf shall use two types of HOHs to enable access to CanDrv:

- Hardware Transmit Handle (HTH) and
- Hardware Receive Handle (HRH).

⌋

[SWS_CANIF_00291] ⌈Definition of HRH: The HRH shall be a handle referencing a logical Hardware Receive Object of the CAN Controller mailbox.⌋

[SWS_CANIF_00665] ⌈The HRH shall enable CanIf to use *BasicCAN* or a *FullCAN* reception method of the referenced reception unit and to indicate a Received L-SDU to a target upper layer module.⌋

[SWS_CANIF_00663] ⌈If the HRH references a reception unit configured for *BasicCAN reception*, software filtering shall be enabled in CanIf.⌋

[SWS_CANIF_00664] ⌈If multiple HRHs are used, each HRH shall belong at least to a single or fixed group of Rx L-SDU (CanRxPduIds).⌋

The HRH can be configured to receive

- one single CanId (*FullCAN*)
- a group of single CanIds (*BasicCAN*)
- a range/area of CanIds (*BasicCAN*) or
- all CanIds.

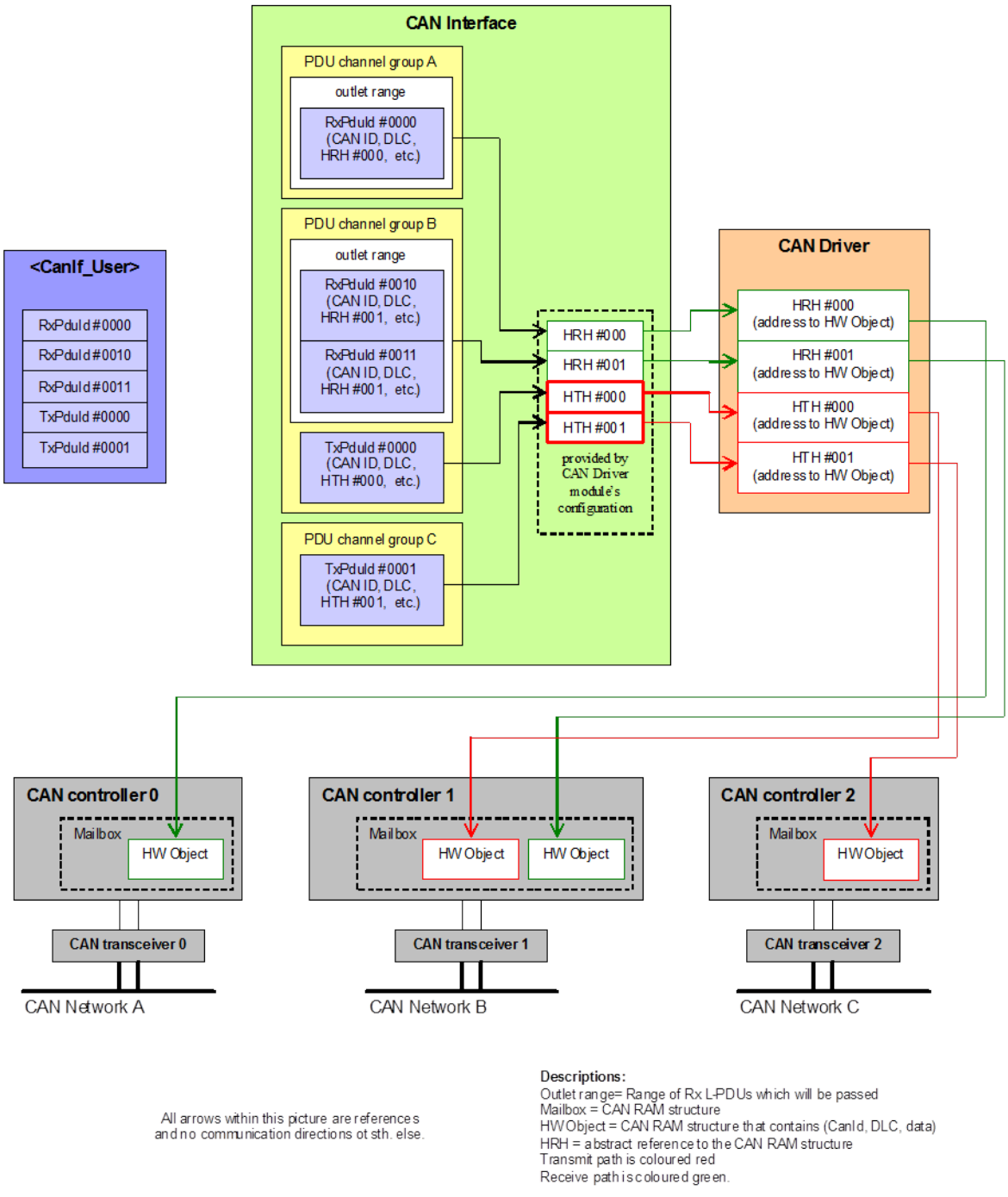


Figure 7.1: Mapping between PDU Ids and HW object handles

[SWS_CANIF_00292] ⌈Definition of HTH: The HTH shall be a handle referencing a logical Hardware Transmit Object of the CAN Controller mailbox.⌋

[SWS_CANIF_00666] ⌈The HTH shall enable CanIf to use *BasicCAN* or *FullCAN* transmission method of the referenced transmission unit and to confirm a transmitted L-SDU to a target upper layer module.⌋

[SWS_CANIF_00466] ⌈Each CanIf Tx L-PDU shall statically be assigned to one CanIfBufferCfg configuration container at configuration time (see CanIfTxPduBufferRef).⌋

Rationale for [SWS_CANIF_00466]: CanIf Tx L-PDUs do not refer HTHs, but CanIfBufferCfg, which in turn do refer HTHs.

[SWS_CANIF_00667] ⌈If multiple HTHs are used, each HTH shall belong to a single or fixed group of Tx L-PDU (CanTxPduIds).⌋

[SWS_CANIF_00115] ⌈CanIf shall be able to use all HRHs and HTHs of one CanDrv as common, single numbering area starting with zero.⌋

The dedicated HRHs and HTHs are derived from the configuration set of CanDrv. The definition of HTH/HRH inside the numbering area and Hardware Objects is up to CanDrv.

7.3 Static L-PDUs

CanIf offers general access to the CAN L-SDU related data for upper layers. Attributes of the following table are represented as configuration parameters and are specified in Chapter 10:

CAN Interface specific attributes	CAN Protocol Control Information (PCI)
Method of SW filtering CanIfPrivateSoftwareFilterType	CAN Identifier (CanId) CanIfTxPduCanId, range of CanIds per *PDU* (see CanIfRxPduCanIdRange), CanIfRxPduCanId, CanIfRxPduCanIdMask
Direction of L-PDU (Tx, Rx) CanIfTxPduId, CanIfRxPduId	Type of CAN Identifier (*StandardCAN*, *ExtendedCAN*) referenced from CanDrv via CanIfHthIdSymRef, CanIfHrhIdSymRef
HTH/HRH of the CAN Controller	Data Length and Data Length Code (*DLC*) CanIfRxPduDataLength
Target ID for the upper layer LSduR	Reference to the PDU data (see [1, Specification of CAN Driver])
Type of Transmit L-PDU (STATIC, DYNAMIC) CanIfTxPduType	
Type of Tx/Rx L-PDU (*FullCAN*, *BasicCAN*) CanIfHthIdSymRef, CanIfHrhIdSymRef	

CanIf supports activation and deactivation of all L-PDUs belonging to one CAN Controller for transmission as well as for reception (see 7.19.2, see CanIf_SetPduMode(), [SWS_CANIF_00008]). For L-PDU mode control refer to Chapter 7.19.

Each L-PDU is associated with an upper layer module in order to ensure correct dispatching during reception, transmission confirmation, and data access. Each upper layer module can use the L-PDUs to serve different CAN Controllers simultaneously.

According to the *PDU* architecture defined for the entire AUTOSAR communication stack (see [10, Layered Software Architecture]), the usage of L-PDUs is split in two different ways:

- For transmission request and transmission/reception polling API the upper layer module uses the L-SDU ID (CanTxPduId/CanRxPduId) defined by CanIf as parameter.
- For all callback APIs, which are invoked by CanIf at upper layer modules, CanIf passes the target PduId defined by each upper layer module as parameter.

The principle is that the caller must use the defined target L-PDU/L-SDU Id of the callee.

If power on initialization is not performed and upper layer performs transmit requests to CanIf, no L-SDUs are transmitted to lower layer and *DET* shall be invoked. Thus, no un-initialized data can be transmitted on the network. Behavior of L-PDU/L-SDU transmitting function is specified in detail in Chapter 8.3.6.

7.4 Dynamic L-PDUs

CanIf shall support the ability to filter incoming messages using the CanIfRxPduCanIdMask. The filtering shall be done by comparing the incoming CanId with the stored CanIfRxPduCanId after applying the CanIfRxPduCanIdMask to both IDs. This should be done after the filtering of regular CanIds without mask, to allow for separate handling of some of the CanIds that fall into the range defined by the mask or a CanId based range.

Additionally, DYNAMIC Tx and Rx L-SDUs shall be supported, where the CanId resides in the MetaData of the L-SDU.

During transmission of dynamic L-SDUs, when a CanIfTxPduCanIdMask is defined, the variable parts of the CanId provided via the MetaData must be merged with the CanId by using this mask. When no CanIfTxPduCanIdMask and no CanIfTxPduCanId are configured, the MetaData shall be used directly as CanId.

During reception of dynamic L-SDUs, the received CanId shall be placed in the L-SDU MetaData. The content of the MetaData is independent of the CanIfRxPduCanIdMask parameter.

[SWS_CANIF_00844] ⌈CanIf shall support dynamic L-PDUs, where the CanId or relevant parts of the CanId are placed in the MetaData of a L-SDU.⌋

7.4.1 Dynamic Transmit L-PDUs

Definition of dynamic Transmit L-PDUs: L-PDUs which allow reconfiguration of the CanId during runtime (CanIfTxPduType) or where the ID or parts thereof are provided as MetaData of the L-SDU.

The Data Length and the pointer to the data buffer are both determined by the upper layer module at call of CanIf_Transmit() and cannot be reconfigured.

The function CanIf_SetDynamicTxId() (see [SWS_CANIF_00189]) reconfigures the CanId of a dynamic L-PDU with CanIfTxPduType.

[SWS_CANIF_00188] ⌈CanIf shall process the two most significant bits of the CanId (see [1, Specification of CAN Driver], definition of Can_IdType [*SWS_Can_00416*]) to determine which type of CanId is used and thus how the dynamic Transmit L-PDU shall be transmitted.⌋

[SWS_CANIF_00673] ⌈The CanIf shall guarantee data consistency of the CanId in case of running function CanIf_SetDynamicTxId(). This service may be interrupted by a *pre-emptive* call of CanIf_Transmit() affecting the same L-PDU, see [SWS_CANIF_00064].⌋

[SWS_CANIF_00855] ⌈If CanIfTxPduCanIdMask and CanIfTxPduCanId are omitted, the CanId is directly taken from the MetaData.⌋

[SWS_CANIF_00856] ⌈CanIfTxPduCanIdMask shall be ignored when meta data configuration does not contain CAN_ID_32 for this L-SDU.⌋

[SWS_CANIF_00854] ⌈If the MetaDataItem CAN_ID_32, CanIfTxPduCanIdMask and CanIfTxPduCanId are available, CanIfTxPduCanIdMask defines the bits in CanIfTxPduCanId (including information about the CAN ID type and CAN standard encoded in the Can_IdType) that shall appear in the actual CanId, the other bits are taken from the MetaData.⌋

Note: The resulting ID could be calculated in the following way: (CanIfTxPduCanId & CanIfTxPduCanIdMask) | (<dynamic ID parts> & ~CanIfTxPduCanIdMask)

[SWS_CANIF_00857] ⌈CanIf_Init() (see [SWS_CANIF_00085]) initializes the CanIds of the dynamic Transmit L-PDUs with CanIfTxPduType to the value configured via CanIfTxPduCanId.⌋

7.4.2 Dynamic receive L-PDUs

Definition of dynamic Receive L-PDUs: L-PDUs that correspond to a set of CanIds, where the actually received CanId is provided to upper layers as part of the PDU data.

[SWS_CANIF_00847] ⌈Configuration shall ensure that dynamic Receive L-PDUs use an ID range or a mask and that the MetaDataItem CAN_ID_32 is configured for the L-SDU. Besides, the software filtering must be enabled for these L-SDUs.⌋

[SWS_CANIF_00848] ⌈Upon reception of a dynamic L-SDU, CanIf shall place the CanId in the MetaDataItem of type CAN_ID_32.⌋

7.5 Physical channel view

A physical channel is linked with one CAN Controller and one CAN Transceiver, whereas one or multiple physical channels may be connected to a single network.

The CanIf provides services to control all CAN devices like CAN Controllers and CAN Transceivers of all supported ECU's CAN channel. Those APIs are used by the CanSm to provide a network view to the ComM (see [3]) used to perform *wake up* and *sleep* request for all physical channels connected to a single network.

The CanIf passes status information provided by the CanDrv and CanTrcv separately for each physical channel as status information for the CanSm (<User_Controller-BusOff>(), refer to [SWS_CANIF_00014]).

[SWS_CANIF_00653] ⌈The CanIf shall provide a ControllerId, which abstracts from the different Controllers of the different CanDrv instances. The range of the ControllerIds within the CanIf shall start with '0'. It shall be configurable via CanIf-CtrlId.⌋

Example:

CanIf	CanDrv A	CanDrv B
ControllerId 0	Controller 0	
ControllerId 1	Controller 1	
ControllerId 2		Controller 0

[SWS_CANIF_00655] ⌈The CanIf shall provide a TransceiverId, which abstracts from the different Transceivers of the different CanTrcv instances. The range of the TransceiverIds within the CanIf shall start with '0'. It shall be configurable via CanIfTrcvId.⌋

Example:

CanIf	CanDrv A	CanDrv B
TransceiverId 0	Transceiver 0	
TransceiverId 1	Transceiver 1	
TransceiverId 2		Transceiver 0

During the notification process the CanIf maps the original CAN Controller or CAN Transceiver parameter from the Driver module to the CanSm. This mapping is done as the referenced CAN Controller or CAN Transceiver parameters are configured with the abstracted CanIf parameters `ControllerId` or `TransceiverId`.

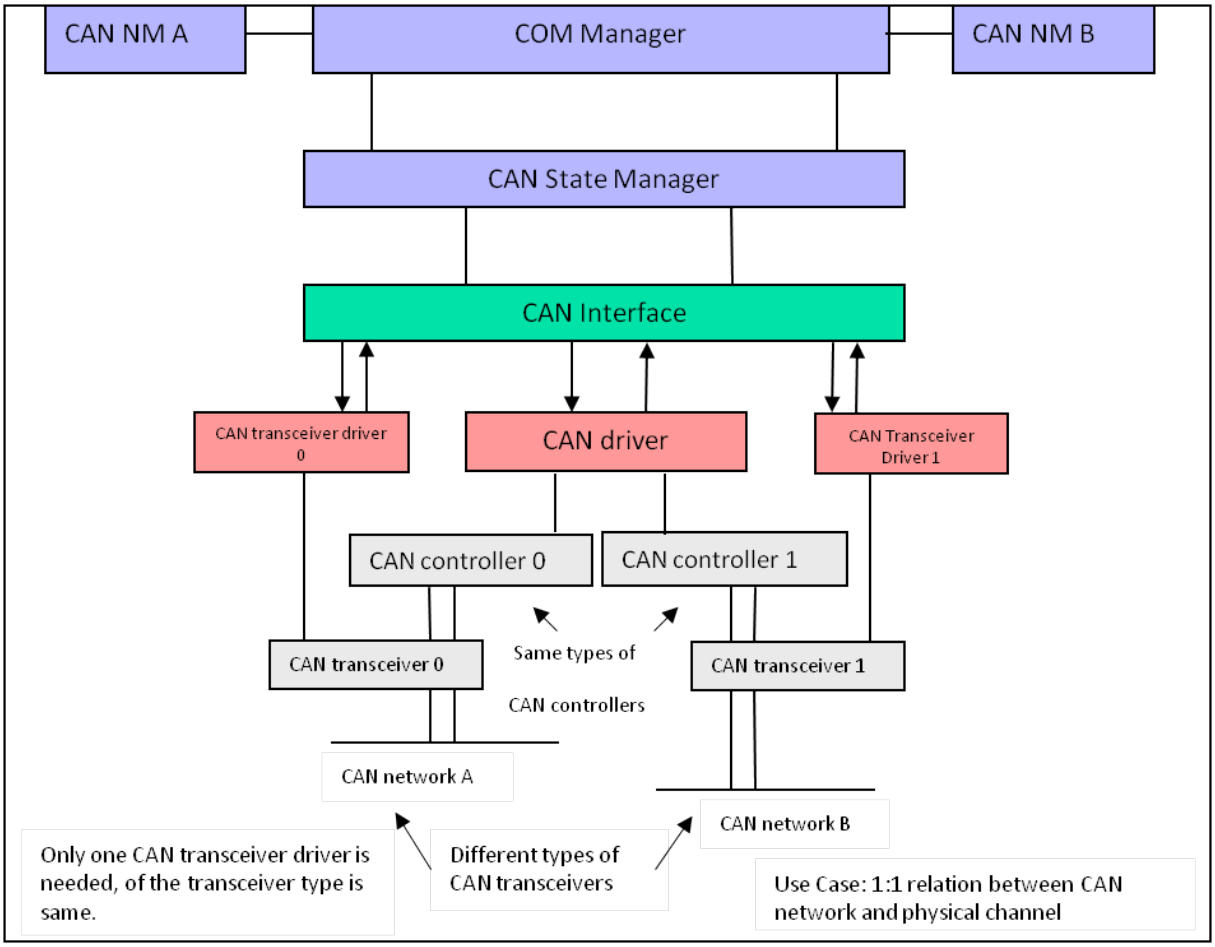


Figure 7.2: Physical channel view definition example A

The CanIf supports multiple physical CAN channels. These have to be distinguished by the CanSm for network control. The CanIf API provides request and read control for multiple underlying physical CAN channels.

Moreover the CanIf does not distinguish between dedicated types of CAN physical layers (i.e. *Low-Speed CAN* or *High-Speed CAN*), to which one or multiple CAN Controllers are connected.

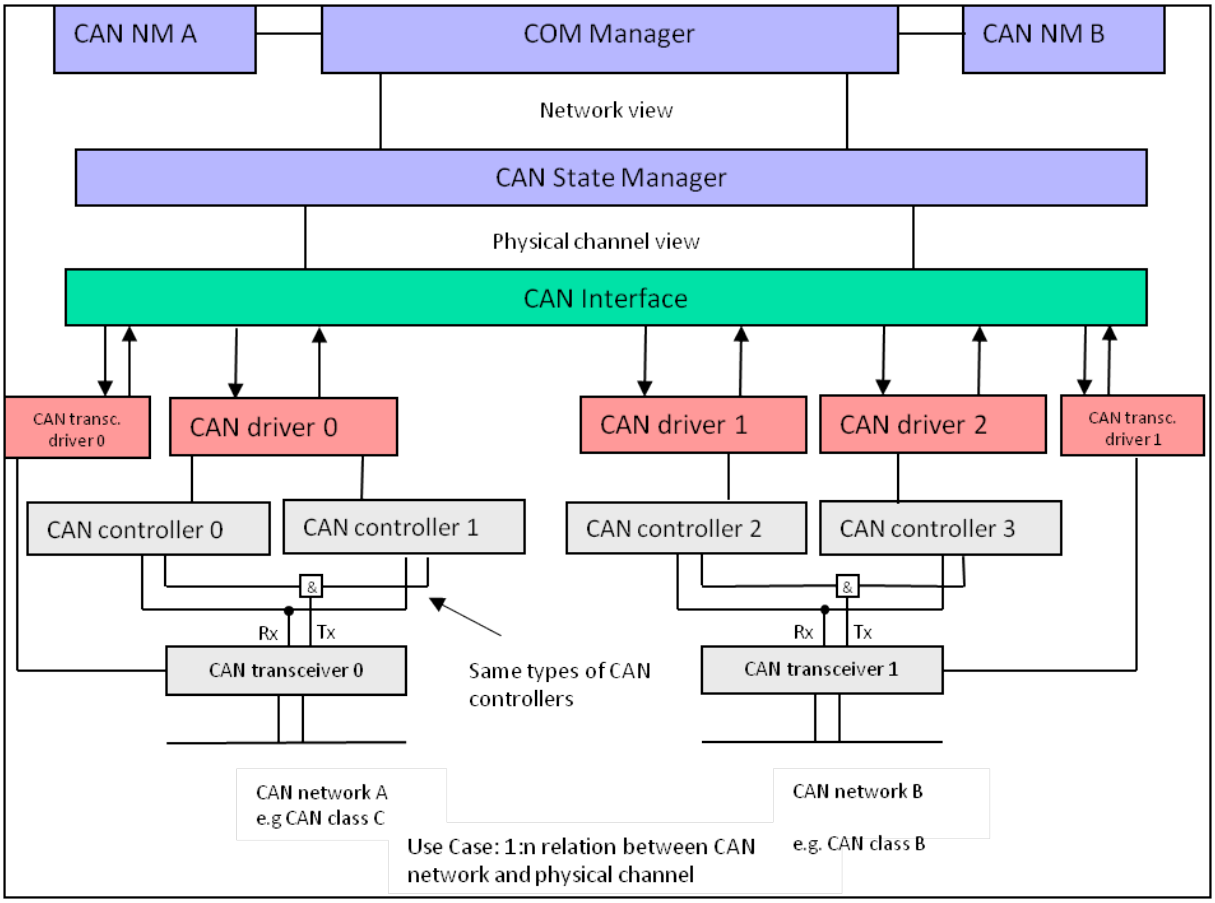


Figure 7.3: Physical channel view definition example B

7.6 CAN Hardware Unit

The CAN Hardware Unit combines one or multiple CAN Controller modules of the same type, which may be located on-chip or as external standalone devices. Each CAN Hardware Unit is served by the corresponding `CanDrv`.

If different types of `CAN Controllers` are used, also different types of `CanDrvs` have to be applied with a unified API to `CanIf`. `CanIf` collects information about number and types of `CAN Controllers` and their `Hardware Objects` at configuration time. This allows transparent and hardware independent access to the `CAN Controllers` from upper layer modules using `HOHs` (refer to Chapter 7.2 “Hardware object handles” and Chapter 7.23 “Multiple CAN Driver support”).

Figure 7.4 shows a CAN Hardware Unit consisting of two CAN Controllers of the same type connected to two physical channels:

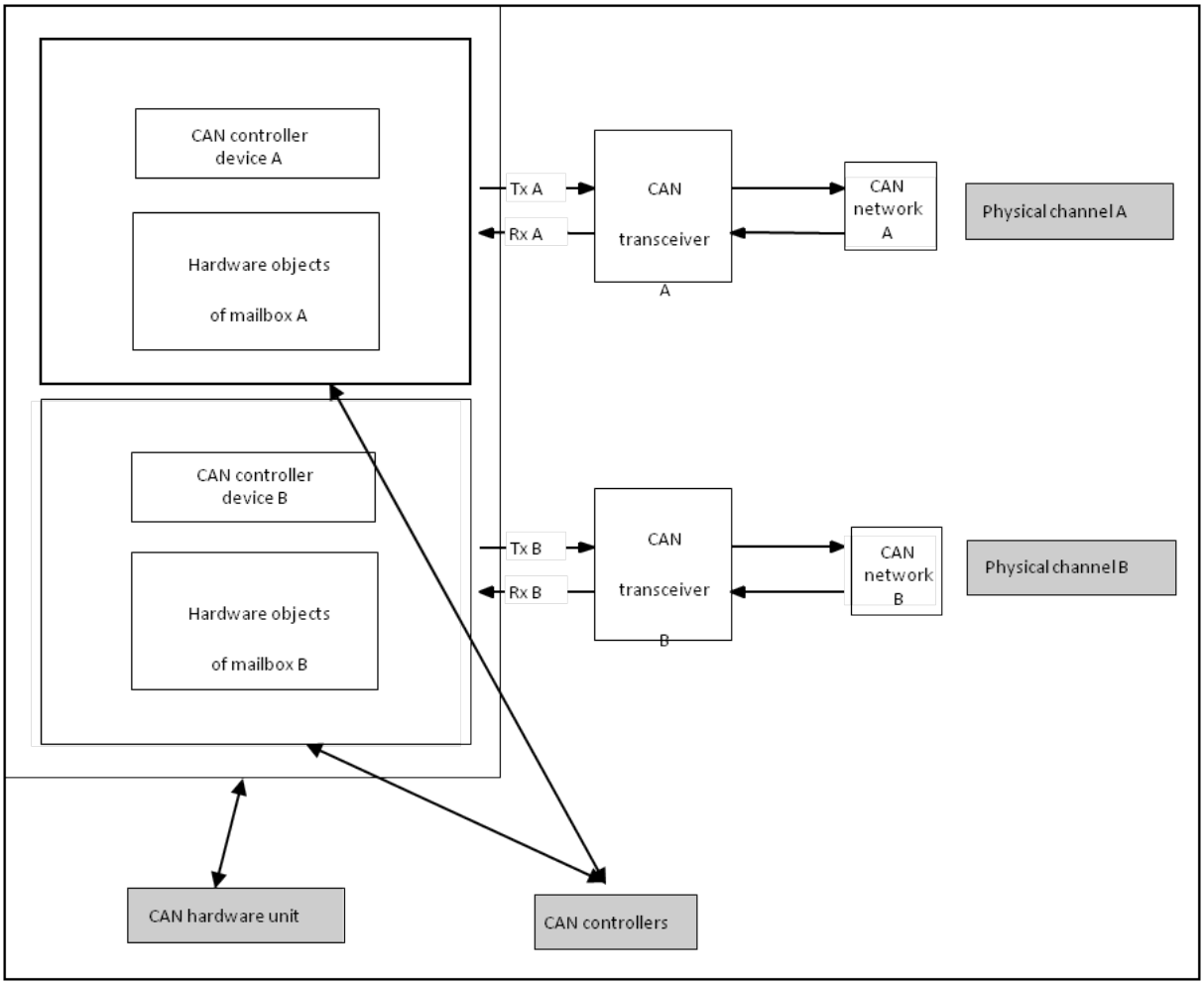


Figure 7.4: Typical CAN Hardware Unit

7.7 BasicCAN and FullCAN reception

This chapter only applies to `CAN 2.0` and `CAN FD`.

CAN XL hardware has an entirely different approach for accessing the frames using descriptors and queues, similar to Ethernet.

`CanIf` distinguishes between *BasicCAN* and *FullCAN* handling for activation of software acceptance filtering.

A CAN mailbox (`Hardware Object`) for *FullCAN* operation only enables transmission or reception of single `CanIds`. Accordingly, *BasicCAN* operation of one `Hardware Object` enables to transmit or receive a range of `CanIds`.

A `Hardware Receive Object` for configured *BasicCAN* reception is able to receive a range of `CanIds`, which pass its hardware acceptance filter. This range may exceed the list of predefined `Rx L-PDUs` to be received by this `HRH`. Therefore, `CanIf` subsequently shall execute software filtering to pass only the predefined list of `Rx L-PDUs`

to the corresponding upper layer modules. For more details please refer to Chapter 7.20 "Software receive filter".

[SWS_CANIF_00467] ⌈CanIf shall configure and store an order on HTHs and HRHs for all HOHs derived from the configuration containers CanIfHthCfg and CanIfHrhCfg.⌋

[SWS_CANIF_00468] ⌈CanIf shall reference a hardware acceptance filter for each HOH derived from the configuration parameters CanIfHthIdSymRef and CanIfHrhIdSymRef.⌋

The main difference between *BasicCAN* and *FullCAN* operation is in the need of a software acceptance filtering mechanism (see Chapter 7.20 "Software receive filter").

[SWS_CANIF_00469] ⌈CanIf shall give the possibility to configure and store a software acceptance filter for each HRH of type *BasicCAN* configured by parameter CanIfHrhSoftwareFilter.⌋

[SWS_CANIF_00211] ⌈CanIf shall execute the software acceptance filter from [SWS_CANIF_00469] for the HRH passed by callback function CanIf_RxIndication().⌋

BasicCAN and *FullCAN* objects may coexist in a single configuration setup. Multiple *BasicCAN* and *FullCAN* receive objects can be used, if provided by the underlying CAN Controllers.

[SWS_CANIF_00877] ⌈If CanIf receives an L-PDU (see CanIf_RxIndication()), it shall compare the masked CanIfRxPduCanId with the passed Mailbox->CanId to select the correct reception L-SDU configured in CanIfRxPduCfg. The mask is defined by CanIfRxPduCanIdMask in both cases.⌋

[SWS_CANIF_00281] ⌈CanIf shall accept and handle *StandardCAN IDs* and *ExtendedCAN IDs* on the same Physical Channel (= mixed mode operation).⌋

In a mixed mode operation *Standard CAN IDs* and *Extended CAN IDs* can be used mixed at the same time on the same CAN network. Mixed mode operation can be accomplished, if the *BasicCAN*/*FullCAN* Hardware Objects have been configured separately for either *StandardCAN* or *ExtendedCAN* operation. In case of mixed mode operation the software acceptance filter algorithm (see Chapter 7.20 "Software receive filter") must be able to deal with both type of CanIds.

7.8 Initialization

The EcuM calls the CanIf's function CanIf_Init() for initialization of the entire CanIf (see [SWS_CANIF_00001]). All global variables and data structures are initialized

including flags and buffers during the initialization process. The EcuM executes initialization of CanDrvs and CanTrcvs separately by call of their corresponding initialization services (refer to [1] and [2, Specification of CAN Transceiver Driver]).

The CanIf expects that the CAN Controller remains in *STOPPED* mode like after power-on reset after the initialization process has been completed. In this mode the CanIf and CanDrv are neither able to transmit nor receive CAN L-PDUs (see [SWS_CANIF_00001]).

If re-initialization of the entire CAN modules during runtime is required, the EcuM shall invoke the CanSm (see [3]) to initiate the required state transitions of the CAN Controller by call of CAN Interface module's API service CanIf_SetControllerMode(). The CanIf maps the calls from CanSm to calls of the respective CanDrvs (see Chapter 8.5.3).

7.9 Transmit request

CanIf's transmit request function CanIf_Transmit() ([SWS_CANIF_00005]) is a common interface for upper layers to transmit L-PDUs on the CAN network. The upper communication layer modules initiate the transmission only via CanIf's services without direct access to CanDrv. The initiated Transmit Request is successfully completed, if CanDrv could write the L-PDU data into the CAN hardware transmit object.

Upper layer modules use the API service CanIf_Transmit() to initiate a transmit request (refer to Chapter 8.3.6 "CanIf_Transmit").

CanIf performs following actions for L-PDU transmission at call of the service CanIf_Transmit():

- Check, initialization status of CanIf
- Identify CanDrv (only if multiple CanDrvs are used)
- Determine HTH for access to the CAN hardware transmit object
- Call Can_Write() or CanXL_Write() of CanDrv

The transmission is successfully completed, if the transmit request service CanIf_Transmit() returns E_OK.

[SWS_CANIF_00382] ⌈If an L-PDU is requested to be transmitted via a PDU channel mode, which equals CANIF_OFFLINE, the CanIf shall report the runtime error code CANIF_E_STOPPED to the Det_ReportRuntimeError() service of the *DET* and CanIf_Transmit() shall return E_NOT_OK.⌋

Note for [SWS_CANIF_00382]: See Chapter 7.19.2 "PDU channel modes".

If the call of Can_Write() returns with CAN_BUSY, please refer to Chapter 7.12 "Transmit confirmation" for further details.

7.10 Transmit data flow

The Transmit Request service CanIf_Transmit() is based on L-PDUs. The access to the L-SDU specific data is organized by the following parameters:

- Transmit L-PDU => L-SDU ID
- Reference to a data structure, which contains L-SDU related data: Pointer to the L-SDU, pointer to the MetaData and L-SDU length.

The reference to the L-SDU data structure is used as a parameter in several CanIf's API services, e.g. CanIf_Transmit() or the callback service LSduR_CanIfRx-Indication(). In case the L-PDU is configured for triggered transmission, the L-SDU pointer is a null pointer.

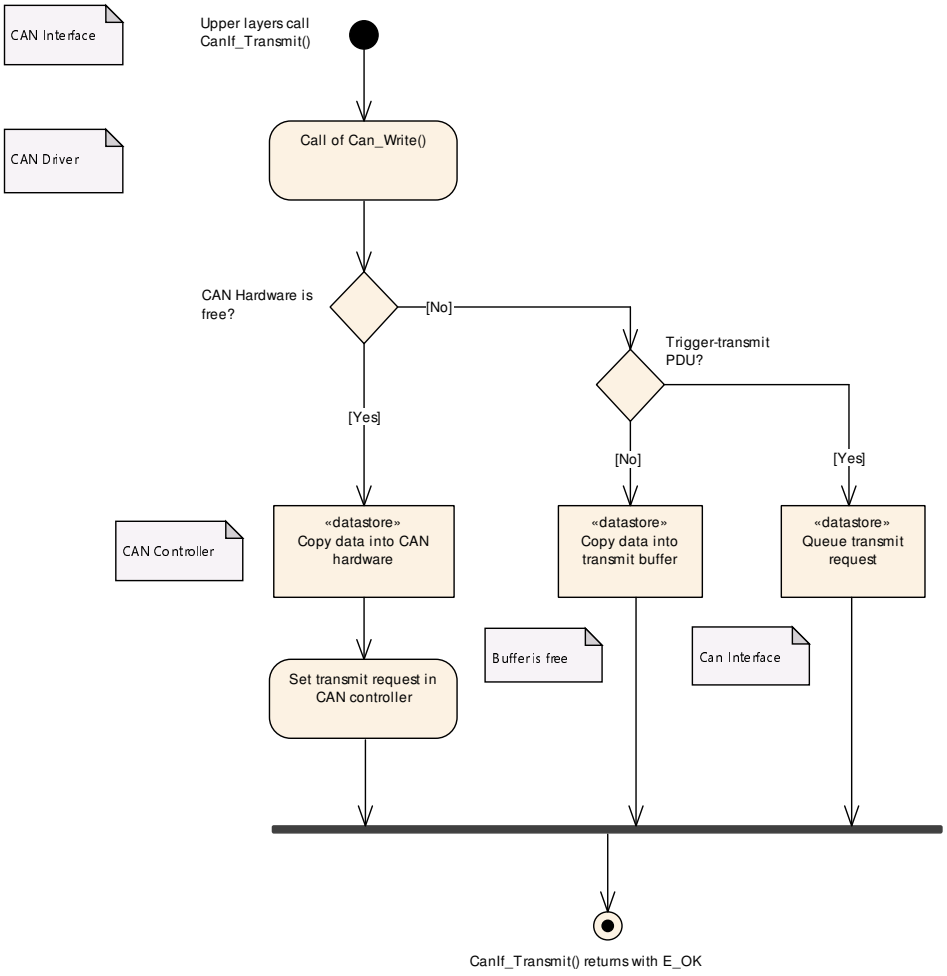


Figure 7.5: Transmit data flow

CanIf stores information about the available hardware objects configured for transmission purposes. The function CanIf_Transmit() maps the TxPduId to the corresponding HTH and calls the function Can_Write() or CanXL_Write().

[SWS_CANIF_00904] ⌈If Bus Mirroring is enabled globally (see CanIfBusMirroringSupport) and has been activated with a call to CanIf_EnableBusMirroring() for a CAN Controller, the CanIf shall store the content of each frame before it is transmitted on that controller with Can_Write().⌋

Note: The frame content should only be provided to the Bus Mirroring module when it was actually sent. Therefore, the content has to be stored so that it can be provided to the Bus Mirroring module from within the CanIf_TxConfirmation().

7.11 Transmit buffering

This chapter only applies to CAN 2.0 and CAN FD. Using transmit buffering for CAN XL within CanIf is not supported.

Note: In case the buffers provided by the hardware are not sufficient, the CAN XL Driver might implement software buffers.

7.11.1 General behavior

At the scope of CanIf the transmit process starts with the call of CanIf_Transmit() and it ends with invocation of upper layer module's callback service LSduR_CanIfTxConfirmation(). During the transmit process CanIf, CanDrv and the CAN Mailbox altogether shall store the L-PDU to be transmitted only once at a single location. Depending on the transmit method, these are:

- The CAN hardware transmit object or
- The Transmit L-PDU Buffer inside CanIf, if transmit buffering is enabled.

For triggered transmission, CanIf only has to store the transmit request for the given L-PDU but not its data. The data is fetched just in time by means of the trigger transmit function when the HTH is free (again). A single Tx L-PDU, requested for transmission, shall never be stored twice. This behavior corresponds to the usual way of periodic communication on the CAN network.

If transmit buffering is enabled, CanIf will store a Tx L-PDU in a CanIf Transmit L-PDU Buffer (CanIfBufferCfg), if it is rejected by CanDrv at Transmit Request.

Basically, the overall buffer in CanIf for buffering Tx L-PDUs consits of one or multiple CanIfBufferCfg (see CanIfBufferCfg). Whereas each CanIfBufferCfg is assigned to one dedicated CanIfBufferHthRef (see CanIfBufferHthRef) and can be configured to buffer one or multiple Tx L-PDUs. But as already mentioned above only one instance per Tx L-PDU can be buffered in the overall amount of CanIfBufferCfg.

The behavior of CanIf during L-PDU transmission differs whether transmit buffering is enabled in the configuration setup for the corresponding Tx L-PDU, or not. If transmit buffering is disabled and a transmit request to CanDrv fails (CAN Controller mailbox is in use, *BasicCAN*), the L-PDU is not copied to the CAN Controller's mailbox and CanIf_Transmit() returns the value E_NOT_OK. If transmit buffering is enabled and a transmit request to CanDrv fails, depending on the CanIfTxBuffer configuration the L-PDU can be stored in a CanIfTxBuffer. In this case the API CanIf_Transmit() returns the value E_OK although the transmission could not be performed. In this case CanIf takes care of the outstanding transmission of the L-PDU via CanIf_TxConfirmation() callback and the upper layer doesn't have to retry the transmit request.

The number of available transmit CanIf Tx L-PDU Buffers can be configured completely independent from the number of used Transmit L-PDUs defined in the CAN network description file for this ECU.

As per [SWS_CANIF_00835] a Tx L-PDU refers HTHs via the CanIfBufferCfg configuration container (see CanIfBufferCfg). This is valid if transmit buffering is not needed as well. In this case, the buffer size (see CanIfBufferSize) of the CanIfBufferCfg has to be set to *0*. Then CanIfBufferCfg configuration container is only used to refer a HTH.

7.11.2 Buffer characteristics

CanIfTxPduBufferRef, CanIfBufferCfg, CanIfBufferHthRef and CanIfBufferSize describe the possible CanIfBufferCfg configurations.

7.11.2.1 Storage of L-PDUs in the transmit L-PDU buffer

CanIf tries to store a new Transmit L-PDU or its Transmit Request in the Transmit L-PDU Buffer only, if CanDrv return CAN_BUSY during a call of Can_Write() (see [SWS_CANIF_00381]).

[SWS_CANIF_00063] ⌈The CanIf shall support buffering of a CAN L-PDU for *BasicCAN* transmission in the CanIf, if parameter CanIfPublicTxBuffering (see CanIfPublicTxBuffering) is enabled.⌋

[SWS_CANIF_00849] ⌈For dynamic Transmit L-PDUs, also the CanId has to be stored in the CanIfTxBuffer.⌋

[SWS_CANIF_00381] ⌈If transmit buffering is enabled (see [SWS_CANIF_00063]) and if the call of Can_Write() for a PDU configured for direct transmission returns with CAN_BUSY, CanIf shall check if it is possible to buffer the CanIf Tx L-PDU, which was requested to be transmitted via Can_Write() in a CanIfTxBuffer.⌋

When the call of Can_Write() returns with CAN_BUSY, CanDrv has rejected the requested transmission of the L-PDU (see [1]) because there is no free hardware object available at time of the transmit request (Tx request).

[SWS_CANIF_00881] ⌈If transmit buffering is enabled (see [SWS_CANIF_00063]) and if the call of Can_Write() for a PDU configured for triggered transmission returns with CAN_BUSY, CanIf shall check if it is possible to buffer the Transmit Request, which was requested to be transmitted via Can_Write() in a CanIfTxBuffer.⌋

[SWS_CANIF_00835] ⌈When CanIf checks whether it is possible to buffer a CanIf Tx L-PDU or a Transmit Request (see [SWS_CANIF_00381], [SWS_CANIF_00881]), this shall only be possible, if the CanIf Tx L-PDU is assigned (see CanIfTxPduBufferRef) to a CanIfBufferCfg (see CanIfBufferCfg), which is configured with a buffer size (see CanIfBufferSize) bigger than zero.⌋

The buffer size of any CanIfTxBuffer is only configurable bigger than zero, if transmit buffering is enabled. Additionally the buffer size of a single CanIfTxBuffer is only configurable bigger than zero if the CanIfTxBuffer is not assigned to a FullCAN HTH (see CanIfBufferSize).

[SWS_CANIF_00836] ⌈If it is possible to buffer a CanIf Tx L-PDU or a Transmit Request, because the buffer size of the assigned CanIfTxBuffer is bigger than zero (see [SWS_CANIF_00835]), CanIf shall buffer a CanIf Tx L-PDU or the Transmit Request in a free buffer element of the assigned CanIfTxBuffer, if the CanIf Tx L-PDU or the Transmit Request is not already buffered in the CanIfTxBuffer.⌋

[SWS_CANIF_00068] ⌈If it is possible to buffer a CanIf Tx L-PDU or a Transmit Request, because the buffer size of the assigned CanIfTxBuffer is bigger than zero (see [SWS_CANIF_00835]), CanIf shall overwrite direct transmitted CanIf Tx L-PDU in the assigned CanIfTxBuffer, if the CanIf Tx L-PDU is already buffered in the CanIfTxBuffer when Can_Write() returns CAN_BUSY.⌋

Note: There is nothing to do for already stored Transmit Requests (see [SWS_CANIF_00068]) due to the fact the data will be catched by CanDrv directly (using CanIf_TriggerTransmit()). Therefore, the latest data will be sent automatically.

If the order of various transmit requests of different L-PDUs shall be kept, transmit requests of upper layer modules must be connected to previous transmit confirmation notifications. This means that a subsequent L-PDU is requested for transmission by the upper layer modules only, if the transmit confirmation of the previous one was notified by CanIf.

Note: Additionally the order of transmit requests can differ depending on the number of configured hardware transmit objects.

[SWS_CANIF_00837] ⌈If the buffer size is greater zero, all buffer elements are busy and CanIf_Transmit() is called with a new L-PDU (no other instance of the same

L-PDU is already stored in the buffer), then the new L-PDU or its Transmit Request shall not be stored and CanIf_Transmit() shall return E_NOT_OK.⌋

7.11.2.2 Clearance of transmit L-PDU buffers

[SWS_CANIF_00386] ⌈CanIf shall evaluate during transmit confirmation (see [SWS_CANIF_00007]), whether pending CanIf Tx L-PDUs or Transmit Requests are stored within the CanIfTxBuffers, which are assigned to the new free Hardware Transmit Object (see [SWS_CANIF_00466]).⌋

[SWS_CANIF_00668] ⌈If pending CanIf Tx L-PDUs or Transmit Requests are available in the CanIfTxBuffers as per [SWS_CANIF_00386], then CanIf shall call Can_Write() for that pending CanIf Tx L-PDU or Transmit Requests (of the one assigned to the new Hardware Transmit Object) with the highest priority (see [SWS_CANIF_00070]).⌋

[SWS_CANIF_00070] ⌈CanIf shall transmit L-PDUs or Transmit Requests stored in the Transmit L-PDU Buffers in priority order (see [11, ISO 11898-1:2015]) per each HTH. CanIf shall not differentiate between L-PDUs and Transmit Requests.⌋

[SWS_CANIF_00183] ⌈When CanIf calls the function Can_Write() for prioritized L-PDUs and Transmit Requests stored in CanIfTxBuffer and the return value of Can_Write() is E_OK, then CanIf shall remove this L-PDU or Transmit Request from the Transmit L-PDU Buffer immediately, before the transmit confirmation returns.⌋

The behavior specified in [SWS_CANIF_00183] simplifies the choice of the new transmit L-PDU stored in the Transmit L-PDU Buffer.

7.11.2.3 Initialization of transmit L-PDU buffers

[SWS_CANIF_00387] ⌈When function CanIf_Init() is called, CanIf shall initialize every Transmit L-PDU Buffer assigned to CanIf.⌋

The requirement [SWS_CANIF_00387] is necessary to prevent transmission of old data after restart of the CAN Controller.

7.11.3 Data integrity of transmit L-PDU buffers

[SWS_CANIF_00033] ⌈CanIf shall protect against concurrent access to Transmit L-PDU Buffers for transmit L-PDUs and Transmit Requests.⌋

This may be realized by using exclusive areas defined within the *BSW Scheduler*. These exclusive areas can e.g. configured, that all interrupts will be disabled while the exclusive area is entered. The corresponding services from the *BSW Scheduler* module are SchM_Enter_CanIf() and SchM_Exit_CanIf().

Rationale: for [SWS_CANIF_00033]: pre-emptive accesses to the Transmit L-PDU Buffer cannot always be avoided. Such Transmit L-PDU Buffer access like storing a new L-PDU or removing transmitted L-PDU may occur preemptively.

7.12 Transmit confirmation

If a previous transmit request is completed successfully, CanDrv notifies it to CanIf by the call of CanIf_TxConfirmation() ([SWS_CANIF_00007]).

[SWS_CANIF_00905] ⌈If Bus Mirroring is enabled globally (see CanIfBusMirroringSupport) and has been activated with a call to CanIf_EnableBusMirroring() for a CAN Controller, the CanIf shall call Mirror_ReportCanFrame() for each frame transmission on that controller that is confirmed with CanIf_TxConfirmation(), providing the stored content and the actual CAN ID.⌋

[SWS_CANIF_00383] ⌈When callback notification CanIf_TxConfirmation() is called, CanIf shall notify the upper communication layer about the performed transmission by call of CanIf's transmit confirmation service LSduR_CanIfTxConfirmation(E_OK).⌋

An upper communication layer module can be designed or configured in a way, that transmit confirmations can be processed with single or multiple callback services for different L-PDUs or groups of L-PDUs. All that services are called by CanIf at transmit confirmation of the corresponding L-PDU transmission request. The Transmit L-PDU enables to dispatch different confirmation services associated to the target upper layer module. This assignment is made statically during configuration.

[SWS_CANIF_00740] ⌈If CanIfPublicTxConfirmPollingSupport is enabled, CanIf shall buffer the information about a received TxConfirmation per CAN Controller, if the controller mode of that controller is in state CAN_CS_STARTED.⌋

7.13 Receive data flow

According to the AUTOSAR Basic Software Architecture the received data will be evaluated and processed in the upper layer communication stacks (i.e. AUTOSAR COM, CanNm, CanTp, DCM). This means, upper layer modules may neither work with (i.e. change) buffers of CanDrv (Rx) nor do they have access to buffers of CanIf (Tx).

CanIf provides internal buffering in the receive path only if CanIfPublicReadRx-PduDataApi is set to TRUE (refer to Chapter 7.15). Tx buffering is addressed in Chapter 7.11 and dynamic L-PDUs are concerned in Chapter 7.4.

In case of a new reception of an L-PDU CanDrv calls CanIf_RxIndication() or CanIf_XLRxIndication() of CanIf, depending on the type of the L-PDU. The access to the L-PDU specific data is organized by these parameters:

- Hardware Receive Handle (HRH)
- Received CAN Identifier (CanId)
- Received Data Length
- Reference to Received L-PDU

The Received L-PDU is hardware dependent (nibble and byte ordering, access type) and allocated to the lowest layer in the communication system - to CanDrv. HRH serves as a link between CanDrv and the upper layer module using the L-PDU. The HRH identifies one CAN hardware receive object, where a new CAN L-PDU was received.

After the indication of a received L-PDU by CanDrv (CanIf_RxIndication() or CanIf_XLRxIndication() is called) the CanIf shall proceed as described in 7.14 Receive indication. CanIf is not able to recognize, whether CanDrv uses temporary buffering or a direct hardware access. It expects normalized L-PDU data in calls of the CanIf_RxIndication() and CanIf_XLRxIndication().

The CAN hardware receive object is locked until the end of the copy process to the temporary or upper layer module buffer. The hardware object will be immediately released after CanIf_RxIndication() of CanIf returns to avoid loss of data.

In case of CAN XL L-PDUs, the receive buffer will be freed after a call to CanXL_XLReleaseRxBuffer().

CanDrv, CanIf and the upper layer module, which belongs to the received L-PDU, access the same temporary intermediate buffer, which can be located either in the CAN hardware receive object of the CAN Controller or as temporary buffer in CanDrv.

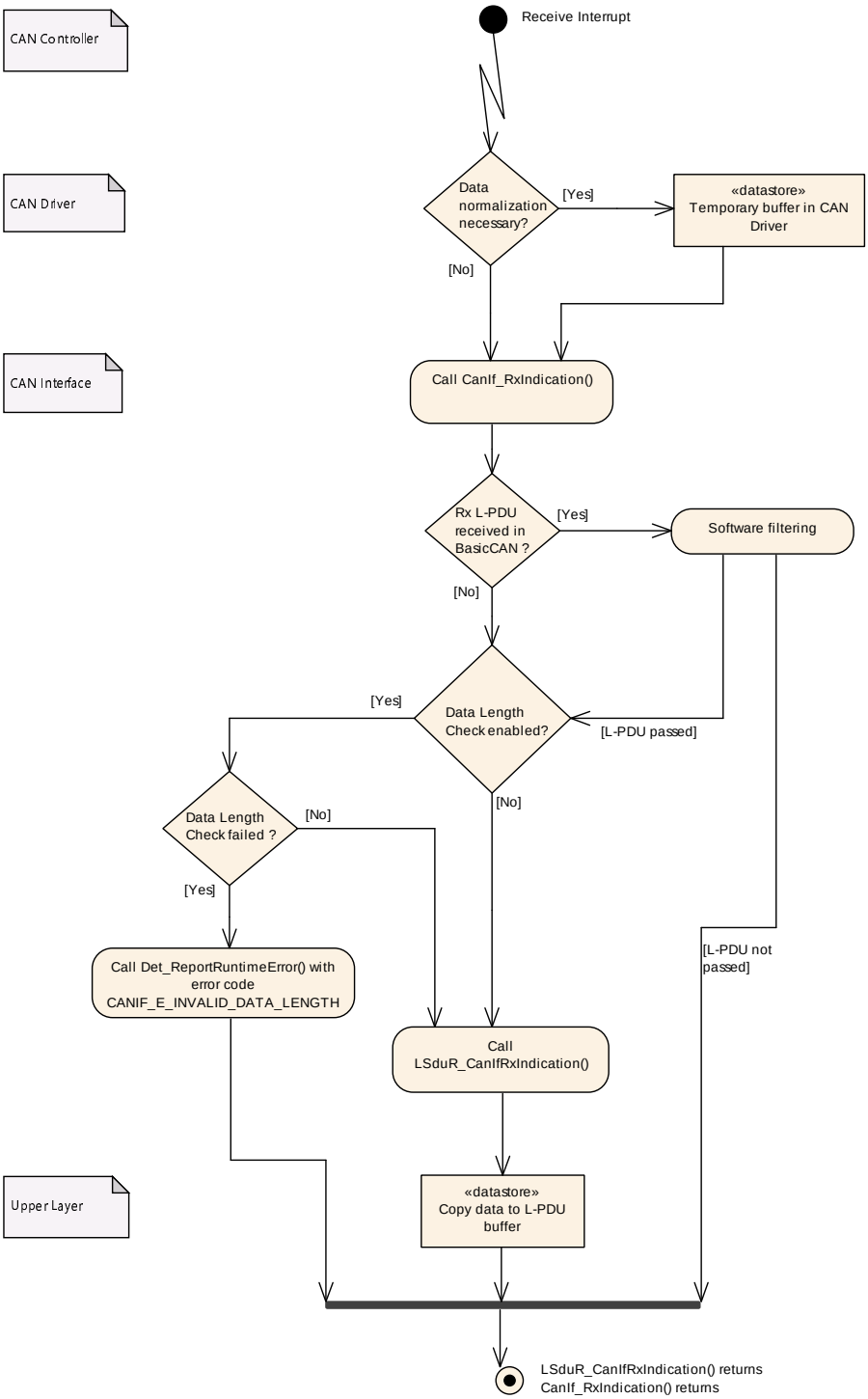


Figure 7.6: Receive data flow

7.14 Receive indication

A call of `CanIf_RxIndication()` or `CanIf_XLRxIndication()` references in its parameters a newly received CAN L-PDU. If one of these functions is called, the CanIf evaluates the CAN L-PDU for acceptance and prepares the `L-SDU` for later access

by the upper communication layers. The CanIf notifies upper layer modules about this asynchronous event using LSduR_CanIfRxIndication(), if configured and if this CAN L-PDU is successfully detected and accepted for further processing. The detailed requirements for this behavior follow here.

[SWS_CANIF_00906] ⌈If Bus Mirroring is enabled globally (see CanIfBusMirroringSupport) and has been activated with a call to CanIf_EnableBusMirroring() for a CAN Controller, the CanIf shall call Mirror_ReportCanFrame() for each frame reception on that controller that is indicated with CanIf_RxIndication().⌋

[SWS_CANIF_00389] ⌈If the function CanIf_RxIndication() or CanIf_XLRxIndication() is called, the CanIf shall process the Software Filtering on the received L-PDU, if configured (see multiplicity of CanIfHrhRangeCfg equals 0..∗). If Software Filtering rejects the received L-PDU, the CanIf shall end this receive indication.⌋

Note for [SWS_CANIF_00389]: See 7.20.

[SWS_CANIF_00390] ⌈If CanIf accepts an L-PDU received via CanIf_RxIndication() or CanIf_XLRxIndication() during Software Filtering (see [SWS_CANIF_00389]), CanIf shall process the Data Length check afterwards, if configured (see CanIfPrivateDataLengthCheck and CanIfRxPduDataLengthCheck).⌋

For further details, please refer to Chapter 7.21 “Data Length Check”.

[SWS_CANIF_00297] ⌈If CanIf has accepted a CAN 2.0 or CAN FD L-PDU received via CanIf_RxIndication() during Data Length Check (see [SWS_CANIF_00390]), CanIf shall copy the number of bytes according to the configured Data Length (see CanIfRxPduDataLength) to the static receive buffer, if configured for that L-PDU (see [SWS_CANIF_00198], CanIfRxPduReadData).⌋

[SWS_CANIF_00851] ⌈If MetaData item CAN_ID_32 is configured for a received CAN 2.0 or CAN FD L-SDU, CanIf shall copy the CAN ID to the MetaDataItem of type CAN_ID_32.⌋

[SWS_CANIF_00967] Support of meta data type TIMETUPLE_TYPE_PTR at reception indication

Status: DRAFT

⌈If MetaData item TIMETUPLE_TYPE_PTR is configured for a received L-SDU, CanIf shall transfer the ingress time stamp of the received L-SDU to the MetaDataItem of type TIMETUPLE_TYPE_PTR.⌋

[SWS_CANIF_00056] Forwarding of CAN 2.0 and CAN FD reception ⌈If CanIf accepts a CAN 2.0 or CAN FD L-PDU received via CanIf_RxIndication() during Data Length Check (see [SWS_CANIF_00390], [SWS_CANIF_00026]), CanIf shall

call the LSduR via LSduR_CanIfRxIndication() based on the arguments to CanIf_RxIndication().⌋

[SWS_CANIF_00973] Forwarding of CAN XL reception ⌈If CanIf accepts a CAN XL L-PDU received via CanIf_XLRxIndication() during Data Length Check (see [SWS_CANIF_00390], [SWS_CANIF_00026]), CanIf shall call the LSduR via LSduR_CanIfRxIndication() based on the arguments to CanIf_RxIndication(). If KeepLocalPduBuffer of the global Pdu referenced by CanIfRxPduRef is configured to FALSE, CanIf shall call CanXL_XLReleaseRxBuffer() immediately afterwards.⌋

[SWS_CANIF_00974] Release hardware reception buffer ⌈When CanIf_ReleaseRxBuffer() is called for a CAN XL PDU for which KeepLocalPduBuffer of the global Pdu is configured to TRUE, the CanIf shall call CanXL_XLReleaseRxBuffer() of the corresponding CAN XL controller. For all other PDUs, any call to CanIf_ReleaseRxBuffer() shall be ignored.⌋

Overview: CanIf performs the following steps at a call of CanIf_RxIndication():

- Disable corresponding PnTxFilter, if it is enabled
- Software Filtering (only BasicCAN), if configured
- Data Length Check, if configured
- buffer received L-SDU if configured
- call upper layer receive indication callback service, if configured.

7.15 Read received data

The read received data API CanIf_ReadRxPduData() (see [SWS_CANIF_00194]) is a common interface for upper layer modules to read CAN L-SDUs recently received from the CAN network. The upper layer modules initiate the receive request only via CanIf services without direct access to CanDrv. The initiated receive request is successfully completed, if CanIf wrote the received L-SDU into the upper layer module I-PDU buffer.

[SWS_CANIF_00198] ⌈If the configuration parameter CanIfPublicReadRxPduDataApi is set to TRUE, CanIf shall store each received L-SDU, at which CanIfRxPduReadData is enabled, into a receive L-SDU buffer. This means that if the configuration parameter CanIfRxPduReadData is set to TRUE, CanIf has to allocate a receive L-SDU buffer for this receive L-SDU.⌋

[SWS_CANIF_00199] ⌈After call of CanIf_RxIndication() and passing of software filtering and Data Length Check, CanIf shall store the received L-SDU in this receive L-SDU buffer. During the call of CanIf_ReadRxPduData() the assigned

receive L-SDU buffer containing a recently received L-SDU, CanIf shall avoid preemptive receive L-SDU buffer access events (refer to [SWS_CANIF_00064]) to that receive L-SDU buffer.⌋

7.16 Read Tx/Rx notification status

In addition to the notification callback functions CanIf provides the API service CanIf_ReadTxNotifStatus() (see [SWS_CANIF_00202]) to read the transmit confirmation status of any transmit L-SDU and the API service CanIf_ReadRxNotifStatus() is provided to read the receive indication status of any receive L-SDU.

CanIf's API services CanIf_ReadTxNotifStatus() (see [SWS_CANIF_00202]) and CanIf_ReadRxNotifStatus() (see [SWS_CANIF_00230]) can be enabled/disabled globally or per L-SDU at pre-compile time configuration using the configuration parameters CanIfPublicReadTxPduNotifyStatusApi, CanIfPublicReadRxPduNotifyStatusApi, CanIfTxPduReadNotifyStatus, and CanIfRxPduReadNotifyStatus.

[SWS_CANIF_00472] ⌈If configuration parameter CanIfPublicReadTxPduNotifyStatusApi is set to TRUE, CanIf shall store the current notification status for each transmit L-SDU.⌋

[SWS_CANIF_00473] ⌈If configuration parameter CanIfPublicReadRxPduNotifyStatusApi is set to TRUE, CanIf shall store the current notification status for each receive L-SDU.⌋

Rationale for [SWS_CANIF_00391] and [SWS_CANIF_00393] respectively [SWS_CANIF_00392] and [SWS_CANIF_00394]: This 'read-and-consume' behavior ensures, that at least one successful transmit or receive event occurred after last call of this service.

7.17 Data integrity

[SWS_CANIF_00064] Shared code shall be reentrant

Upstream requirements: SRS_BSW_00312

⌈CanIf shall protect preemptive events, which access shared resources, that could be changed during CanIf's event handling, against each other.⌋

Rationale: An attempt to update the data in the upper layer module buffers as well as in CanIf's internal buffers has to be done with respect to possible changes done in the context of an interrupt service routine or other preemptive events. Preemptive events probably occur either from preemptive tasks, multiple CAN interrupts, if multiple

physical channels i.e. for gateways are used, or in case of other peripherals or network systems interrupts, which have the needs to transmit and receive L-PDUs on the network.

Handling of shared transmit and receive L-PDU/L-SDU buffers are critical issues for the implementation of CanIf. Therefore CanIf shall ensure data integrity and thus use appropriate mechanisms for access to shared resources like transmission/reception L-PDU/L-SDU buffers. Preemptive events, i.e. transmission and reception event from other CAN Controllers could compromise data integrity by writing into the same L-PDU/L-SDU buffer.

CanIf can e.g. use CanDrv services to enable (Can_EnableControllerInterrupts()) and disable (Can_DisableControllerInterrupts()) CAN interrupts and its notifications at entry and exit of the critical sections separately for each CAN Controller. If there are common resources for multiple CAN Controllers, the entire CAN Interrupts must be locked. These sections must not take a long time in order to prevent serious performance degradation. Thus copying of data, change of static variables, counters and semaphores should be carried out inside these critical sections. It is up to the implementation to use appropriate mechanisms to guarantee data integrity, interrupt ability and reentrancy.

The transmit request API CanIf_Transmit() must be able to operate re-entrant to allow multiple transmit request calls caused by different preemptive events of different L-PDUs/L-SDUs. CanDrv's transmit request API Can_Write() or CanXL_Write() operates re-entrant as well.

7.18 CAN Controller Mode

7.18.1 General Functionality

CanIf provides services for controlling the communication mode of all supported CAN Controllers represented by the underlying CanDrv. This means that all CAN Controllers are controlled by the corresponding provided API services to request and read the current controller mode.

The CAN Controller status may be changed at request of the upper layer by the calling of CanIf_SetControllerMode() service. The request is passed by CanIf via the CanDrv API to the addressed CAN Controller.

The consistent management of all CAN Controllers connected at one CAN network is the task of CanSm. By this way CanSm is responsible to set all CAN Controllers of one CAN network sequentially to sleep mode or to wake them up.

CanIf accepts every state transition request by calling the function CanIf_SetControllerMode() or CanIf_ControllerBusOff(). CanIf does not decide if a requested mode transition of the CAN Controller is valid or not. CanIf only interacts with CanDrv by fetching the current mode and execution of requested mode transitions.

This network related state machine is implemented in CanSm. Refer to [3]. CanIf only stores the requested mode and executes the requested transition.

Hint: As optimisation to avoid frequent requests to CanDrv for internal use the last state indicated by CanIf_ControllerModeIndication() and Can_GetControllerMode() could be stored per controller.

Hint: It has to be regarded that not only CanSm is able to request CAN Controller Mode changes.

7.18.2 CAN Controller Operation Modes

According to the requested operation mode by CanSm, CanIf forwards request CanDrvs.

[SWS_CANIF_00677] ⌈If a controller mode referenced by ControllerId is in state CAN_CS_STOPPED and if the PduIdType parameter in a call of CanIf_Transmit() is assigned to that CAN Controller, then the call of CanIf_Transmit() does not result in a call of Can_Write() or CanXL_Write() and returns E_NOT_OK.⌋

[SWS_CANIF_00485] ⌈If a controller mode referenced by ControllerId enters state CAN_CS_STOPPED, then CanIf shall clear the CanIf transmit buffers assigned to the CAN Controller corresponding.⌋

[SWS_CANIF_00739] ⌈If a controller mode referenced by ControllerId enters state CAN_CS_STOPPED, then CanIf shall inform upper layer modules about failed transmission by calling LSduR_CanIfTxConfirmation(id, E_NOT_OK) for every outstanding TxConfirmation assigned to that CAN Controller. If CanIfPublicTxConfirmPollingSupport is enabled, CanIf shall also clear the information about a TxConfirmation (see [SWS_CANIF_00740]).⌋

Note: This ensures, that for each PDU, which shall be transmitted via CanIf_Transmit(), either a positive or negative LSduR_CanIfTxConfirmation() is called.

[SWS_CANIF_00724] ⌈When callback CanIf_ControllerBusOff(ControllerId) is called, the CanIf shall call CanSM_ControllerBusOff(ControllerId) of the CanSm or a *CDD* (see [SWS_CANIF_00559], [SWS_CANIF_00560]).⌋

Note for [SWS_CANIF_00724]: See Chapter 8.5.3.7 “<User_ControllerModeIndication>”.

[SWS_CANIF_00711] ⌈When callback CanIf_ControllerModeIndication(ControllerId, ControllerMode) is called, CanIf shall call CanSM_ControllerModeIndication(ControllerId, ControllerMode) of the CanSm, Mka_CanControllerModeIndication(ControllerId,

ControllerMode) of the MKA and/or a *CDD* (see [SWS_CANIF_00691], [SWS_CANIF_00972] [SWS_CANIF_00692]).⌋

Note for [SWS_CANIF_00711]: See Chapter 8.5.3.7 "<User_ControllerModeIndication>".

[SWS_CANIF_00712] ⌈When callback CanIf_TrcvModeIndication (Transceiver, TransceiverMode) is called, CanIf shall call CanSM_TransceiverModeIndication(TransceiverId, TransceiverMode) of the CanSm or a *CDD* (see [SWS_CANIF_00697], [SWS_CANIF_00698]).⌋

Note for [SWS_CANIF_00712]: See Chapter 8.5.3.7 "<User_ControllerModeIndication>".

7.18.3 Controller Mode Transitions

The API for state change requests to the CAN Controller behaves in an asynchronous manner with asynchronous notification via callback services.

The real transition to the requested mode occurs asynchronously based on setting of transition requests in the CAN controller hardware, e.g. request for sleep transition CAN_CS_SLEEP. After successful change to e.g. CAN_CS_SLEEP mode CanDrv calls function CanIf_ControllerModeIndication() and CanIf in turn calls function <User_ControllerModeIndication>(). If CAN transitions very fast, CanIf_ControllerModeIndication() can be called during CanIf_SetControllerMode(). This is implementation specific.

Unsuccessful or no mode transitions of the CAN Controllers have to be tracked by upper layer modules. Mode transitions CAN_CS_STARTED and CAN_CS_STOPPED are treated similar.

Upper layer modules of CanIf can poll the current Controller Mode by CanIf_GetControllerMode().

Not all types of CAN Controllers support *Sleep* and *Wake-Up Mode*. These modes are then encapsulated by CanDrv by providing hardware independent operation modes via its interface, which has to be managed by CanIf.

Note: It is possible that during transition from CAN_CS_STOPPED to CAN_CS_SLEEP CAN Controller may indicate a wake-up interrupt to the ECU Integration Code.

CanIf distinguishes between internal initiated CAN controller wake-up request (internal request) and network wake-up request (external request). The internal request is initiated by call of CanIf's function CanIf_SetControllerMode(ControllerId, CAN_CS_STARTED) and it is an internal asynchronous request. The external request is a CAN controller event, which is notified by CanDrv or CanTrcv to the ECU Integration Code. For details see respective UML diagram in the chapter "CAN Wakeup Sequences" of document [12].

7.18.4 Wake-up

The ECU supports wake-up over CAN network, regardless of the used wake-up method (directly about CAN Controller or CAN Transceiver), only if the CAN Controller and CAN Transceiver are set to some kind of "listen for wake-up" mode. This is usually a *Sleep Mode*, where the usual communication is disabled. Only this mode ensures that the CAN Controller is stopped. Thus, the wake-up interrupt can be enabled.

7.18.4.1 Wake-up detection

If *wake-up support* is enabled (see ECUC_CanIf_00843) CanIf is notified by the Integration Code about a detected CAN wake-up by the service CanIf_CheckWakeup() (see CAN Wakeup Sequences of [12]).

In case of a CAN bus "*wake-up*" event the function CanIf_CheckWakeup(WakeupSource) may be called during execution of EcuM_CheckWakeup(WakeupSource) (see wake-up sequence diagrams of EcuM). CanIf in turn checks by configured input reference to EcuMWakeupSource in CanDrvs, which CanDrvs have to be checked. CanIf gets this information via reference CanIfCtrlCanCtrlRef.

The Communication Service, which is called, belongs to the service defined during configuration (see CanIfDispatchCfg). In this way EcuM as well as CanSm are able to change CAN Controller States and to control the system behavior concerning the *BusOff recovery* or *wake-up procedure*.

[SWS_CANIF_00395] ⌈When CanIf_CheckWakeup(EcuM_WakeupSourceType WakeupSource) is invoked, CanIf shall query CanDrvs / CanTrcvs via CanTrcv_CheckWakeup() or Can_CheckWakeup(), which exact CAN hardware device caused the bus wake-up.⌋

Note: It is implementation specific, which controllers and transceivers are queried. CanIf just has to find out the exact CAN hardware device.

[SWS_CANIF_00720] ⌈If at least one function call of Can_CheckWakeup() or CanTrcv_CheckWakeup() returns E_OK to CanIf, then CanIf_CheckWakeup() shall return E_OK.⌋

[SWS_CANIF_00678] ⌈If all calls of Can_CheckWakeup() or CanTrcv_CheckWakeup() return E_NOT_OK to CanIf, then CanIf_CheckWakeup() shall return E_NOT_OK.⌋

7.18.4.2 Wake-up Validation

Note: When a CAN Controller / CAN Transceiver detects a bus wake-up event, then this will be notified to the *ECU State Manager* directly. If such a *wake-up event*

needs to be validated, the EcuM (or a *CDD*) switches on the corresponding CAN Controller (CanIf_SetControllerMode()) and CAN Transceiver (CanIf_SetTrcvMode()) (For more details see chapter 9 of [12]).

Attention: CanIf notifies the upper layer modules about received messages after the *PDU Channel Mode* has been set to CANIF_ONLINE or CANIF_TX_OFFLINE. Thus, it is necessary that the *PDU Channel Mode* is not set to CANIF_ONLINE or CANIF_TX_OFFLINE if wake-up validation is required.

Note: As per [SWS_CAN_00411] and *CAN Controller State Diagram* (see [1]) a direct transition from mode CAN_CS_SLEEP to CAN_CS_STARTED is not allowed.

[SWS_CANIF_00286] ⌈If CanIfPublicWakeupCheckValidSupport equals TRUE, CanIf enables the detection for CAN wake-up validation. Therefore, CanIf stores the event of the first valid call of CanIf_RxIndication() of a CAN Controller which has been set to CAN_CS_STARTED. The first call of CanIf_RxIndication() is valid:

- only for received NM messages if CanIfPublicWakeupCheckValidByNM is TRUE
- for all received messages corresponding to a configured Rx PDU if CanIfPublicWakeupCheckValidByNM is FALSE.

⌋

[SWS_CANIF_00179] ⌈<User_ValidateWakeupEvent>(sources) shall be called during CanIf_CheckValidation(WakeupSource), whereas sources is set to WakeupSource, if the event of the first called CanIf_RxIndication() is stored in CanIf at the corresponding CAN Controller.⌋

Note: The parameter of the function <User_ValidateWakeupEvent>() is of type:

- sources: EcuM_WakeupSourceType (see [12])

[SWS_CANIF_00756] ⌈When controller mode is set to CAN_CS_SLEEP the stored event from previous wake-up (first call of CanIf_RxIndication) shall be cleared (see [SWS_CANIF_00179]).⌋

7.19 PDU channel mode control

7.19.1 PDU channel groups

Each L-PDU is assigned to one dedicated physical CAN channel connected to one CAN Controller and one CAN network. By this way all L-PDUs belonging to one Physical Channel can be controlled on the view of handling logically single L-PDU channel groups. Those logical groups represent all L-PDUs of one ECU connected to one underlying CAN network.

Figure 7.7 below shows one possible usage of L-PDU channel group and its relation to the upper layers and/or networks.

An L-PDU can only be assigned to one channel group.

Typical users like PduR or the Network Management are responsible for controlling the PDU operation modes.

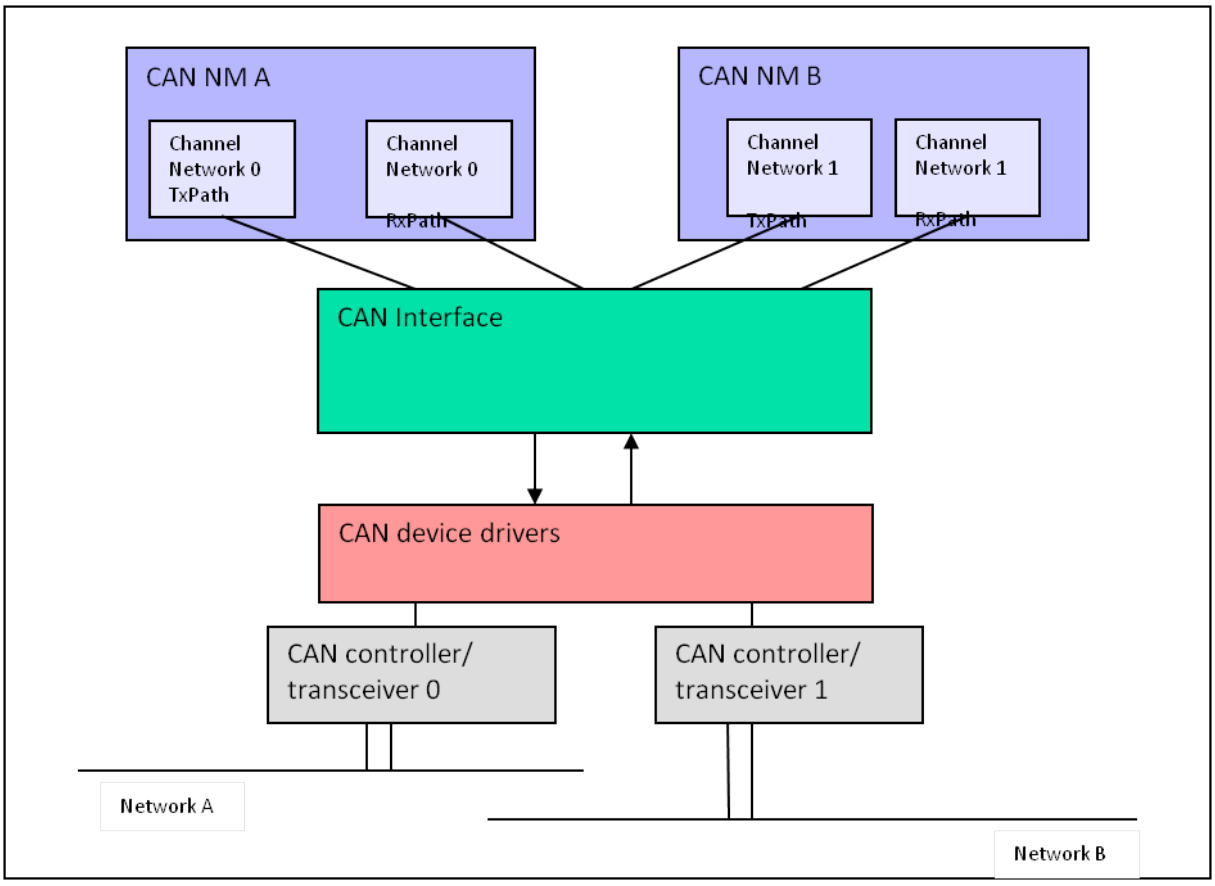


Figure 7.7: Channel PDU groups

7.19.2 PDU channel modes

CanIf provides the services CanIf_SetPduMode() and CanIf_GetPduMode() to prevent the processing of

- all Transmit L-PDUs belonging to one logical channel,
- all Transmit L-PDUs and Receive L-PDUs belonging to one logical channel.

Changing the PDU channel mode is only allowed in case corresponding controller mode equals CAN_CS_STARTED (refer to [SWS_CANIF_00874]).

While CANIF_ONLINE and CANIF_OFFLINE affecting the whole communicatoin the PDU channel modes CANIF_TX_OFFLINE and CANIF_TX_OFFLINE_ACTIVE enable/disable transmission path seperately.

CanIf provides information about the current PDU channel mode via the service CanIf_GetPduMode().

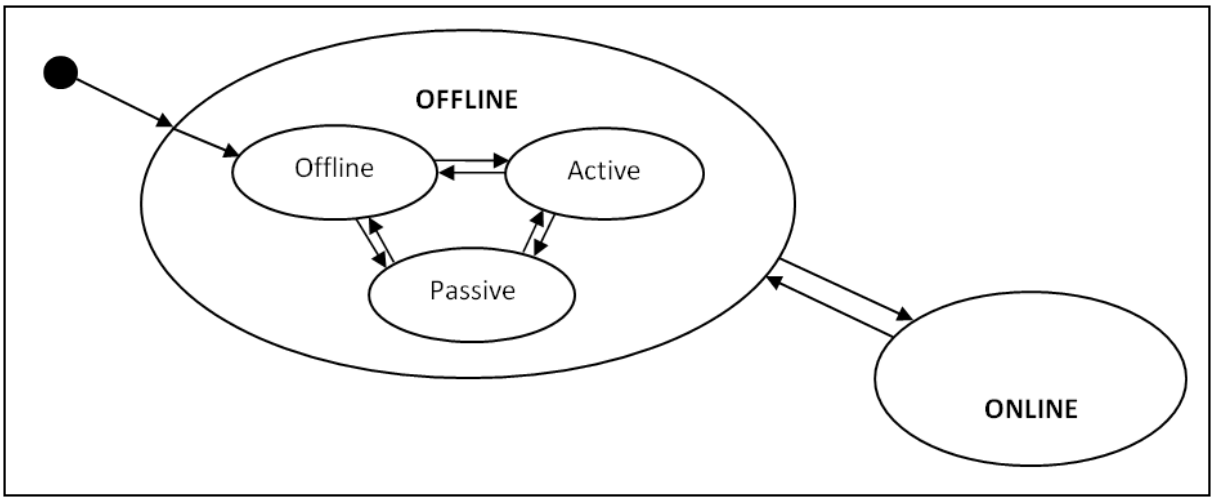


Figure 7.8: PDU channel mode control

Figure 7.8 shows a diagram with possible PDU channel modes. Each L-PDU channel can be in CANIF_OFFLINE (no communication), CANIF_TX_OFFLINE (passive mode => listen without sending), CANIF_TX_OFFLINE_ACTIVE (simulated transmission with listening), and CANIF_ONLINE (full communication). The default state is the CANIF_OFFLINE mode.

7.19.2.1 CANIF_OFFLINE

[SWS_CANIF_00864] ⌈During initialization CanIf shall switch every channel to CANIF_OFFLINE.⌋

[SWS_CANIF_00865] ⌈If CanIf_SetControllerMode(ControllerId, CAN_CS_SLEEP) is called, CanIf shall set the PDU channel mode of the corresponding channel to CANIF_OFFLINE.⌋

[SWS_CANIF_00073] ⌈For Physical Channels switching to CANIF_OFFLINE mode CanIf shall:

- prevent forwarding of transmit requests CanIf_Transmit() of associated L-PDUs to CanDrv (return E_NOT_OK to the calling upper layer modules),
- clear the corresponding CanIf transmit buffers,
- prevent invocation of receive indication callback services of the upper layer modules,
- prevent invocation of transmit confirmation callback services of the upper layer modules.

⌋

[SWS_CANIF_00866] ⌈If `CanIf_SetControllerMode(ControllerId, CAN_CS_STOPPED)` or `CanIf_ControllerBusOff(ControllerId)` is called, `CanIf` shall set the PDU channel mode of the corresponding channel to `CANIF_TX_OFFLINE`.⌋

[SWS_CANIF_00489] ⌈For `Physical Channels` switching to `CANIF_TX_OFFLINE` mode `CanIf` shall:

- prevent forwarding of transmit requests `CanIf_Transmit()` of associated `L-PDUs` to `CanDrv` (return `E_NOT_OK` to the calling upper layer modules),
- clear the corresponding CanIf transmit buffers,
- prevent invocation of transmit confirmation callback services of the upper layer modules.
- enable invocation of receive indication callback services of the upper layer modules.

⌋

The *BusOff* notification is implicitly suppressed in case of `CANIF_OFFLINE` and `CANIF_TX_OFFLINE` due to the fact, that no `L-PDUs` can be transmitted and thus the `CAN Controller` is not able to go in *BusOff* mode by newly requested `L-PDUs` for transmission.

[SWS_CANIF_00118] ⌈If those `Transmit L-PDUs`, which are already waiting for transmission in the `CAN Transmit Hardware Object`, will be transmitted immediately after change to `CANIF_TX_OFFLINE` or `CANIF_OFFLINE` mode and a subsequent *BusOff* event occurs, `CanIf` does not prohibit execution of the *BusOff* notification `<User_ControllerBusOff>(ControllerId)`.⌋

The wake-up notification is not affected concerning PDU channel mode changes.

7.19.2.2 CANIF_ONLINE

[SWS_CANIF_00075] ⌈For `Physical Channels` switching to `CANIF_ONLINE` mode `CanIf` shall:

- enable forwarding of transmit requests `CanIf_Transmit()` of associated `L-PDUs` to `CanDrv`,
- enable invocation of receive indication callback services of the upper layer modules,
- enable invocation of transmit confirmation callback services of the upper layer modules.

⌋

7.19.2.3 CANIF_OFFLINE_ACTIVE

If CanIfTxOfflineActiveSupport = TRUE CanIf provides simulation of successful transmission by CANIF_TX_OFFLINE_ACTIVE mode. This mode is enabled by call of CanIf_SetPduMode(ControllerId, CANIF_TX_OFFLINE_ACTIVE) and only affects the transmission path.

[SWS_CANIF_00072] ⌈For every L-PDU assigned to a channel which is in CANIF_TX_OFFLINE_ACTIVE mode CanIf shall call the transmit confirmation callback services of the upper layer modules immediately instead of buffering or forwarding of the L-PDUs to CanDrv during the call of CanIf_Transmit().⌋

Note: During CANIF_TX_OFFLINE_ACTIVE mode the upper layer has to handle the execution of the transmit confirmations. The transmit confirmation handling is executed immediately at the end of the transmit request (see [SWS_CANIF_00072]).

Rational: This functionality is useful to realize special operating modes (i.e. diagnosis passive mode) to avoid bus traffic without impact to the notification mechanism. This mode is typically used for diagnostic usage.

7.20 Software receive filter

Not all L-PDUs, which may pass the hardware acceptance filter and therefore are successful received in *BasicCAN* Hardware Objects, are defined as Receive L-PDUs and thus needed from the corresponding ECU. CanIf optionally filters out these L-PDUs and prohibits further software processing.

Certain software filter algorithms are provided to optimize software filter runtime. The approach of software filter mechanisms is to find out the corresponding L-PDU from the HRH and CanId currently being processed. After the L-PDU is found, CanIf accepts the reception and enables upper layers to access L-SDU information directly.

The following sections only apply to CAN 2.0 and CAN FD frames, including those received over a tunneled CAN 2.0/CAN FD frame (SDU Type 03h) on CAN XL.

Hint: For content based CAN XL frame (SDU Type 01h) the *PDU_N identifier* which is contained in the Acceptance Field can be used for filtering, while the Priority ID shall not be used for filtering.

[SWS_CANIF_00951] ⌈When a CAN XL frame with SDU Type other than 03h is received, the CanIf shall compare the Acceptance Field, the Priority ID, the SDU Type, and the VCID to the CanIfXLAcceptanceField, CanIfXLPriorityId, CanIfXLSduType, and CanIfXLVcid configured in the CanIfRxPduXLParams of all CanIfRxPduCfgs. If a parameter is not configured in the CanIfRxPduXLParams, it matches any received value.⌋

[SWS_CANIF_00952] ⌈When a CAN XL frame with SDU Type 03h is received, the CanIf shall compare the Priority ID and the VCID to the CanIfXLPriorityId

and CanIfXLVcid configured in the CanIfRxPduXLParams of all CanIfRxPduCfgs where CanIfXLSduType is set to 03h. If a parameter is not configured in the CanIf-RxPduXLParams, it matches any received value.⌋

[SWS_CANIF_00953] ⌈When the CAN XL Params of a received frame with SDU Type other than 03h match the configured CanIfRxPduXLParams of a CanIfRxPduCfg, the corresponding N-PDU shall be forwarded to the upper layer.⌋

[SWS_CANIF_00954] ⌈When the CAN XL Params of a received frame with SDU Type 03h match the configured CanIfRxPduXLParams of a CanIfRxPduCfg, the CAN ID shall be filtered according to the following sections.⌋

[SWS_CANIF_00955] ⌈When the CAN XL Params of a received frame do not match the CanIfRxPduXLParams of any CanIfRxPduCfg, the frame shall be dropped.⌋

[SWS_CANIF_00956] ⌈In case more than one CanIfRxPduCfg matches a received frame according to the rules set up by [SWS_CANIF_00951] and [SWS_CANIF_00952], a matched parameter of CanIfRxPduXLParams shall always be preferred over an unavailable configuration parameter. The matching shall then be executed in the following sequence:

1. SDU Type
2. Acceptance Field
3. VCID
4. Priority ID

A match in a preceding element of this list shall always be preferred over a match in a following element, i.e. the look-up shall be executed based on an ordered list based on this sequence.⌋

With this approach, a stable behavior of the assignment of received frames to L-SDUs (N-PDUs) will be achieved: CanIfRxPduCfgs with a matched SDU Type come first, and the ones with an unavailable SDU Type come last. Within those two blocks, again two blocks will exist for matched Acceptance Field and unconfigured Acceptance Field, and so on down to the Priority ID blocks in the list.

7.20.1 Software filtering concept

The configuration tool handles the information about hardware acceptance filter settings. The most important settings are the number of the L-PDU hardware objects and their range. The outlet range defines, which Receive L-PDUs belongs to each Hardware Receive Object. The following definitions are possible:

- a single Receive L-PDU (*FullCAN* reception),
- a list of Receive L-PDUs or

- one or multiple ranges of Receive L-PDUs can be linked to a Hardware Receive Object (*BasicCAN* reception).

For definition of range reception it is necessary to define at least one Rx L-PDU where the CanId or the complete ID range is inside the defined range.

[SWS_CANIF_00645] ⌈A range of CanIds which shall pass the software receive filter shall either be defined by its upper limit (see CanIfHrhRangeRxPduUpperCanId) and lower limit (see CanIfHrhRangeRxPduLowerCanId) CanId, or by a base ID (see CanIfHrhRangeBaseId) and a mask that defines the relevant bits of the base ID (see CanIfHrhRangeMask).⌋

Note: Software receive filtering is optional (see multiplicity of $0..*$ in CanIfHrhRangeCfg).

[SWS_CANIF_00646] ⌈Each configurable range of CanIds (see [SWS_CANIF_00645]), which shall pass the software receive filter, shall be configurable either for *Standard CAN IDs* or *Extended CAN IDs* via CanIfHrhRangeRxPduRangeCanIdType.⌋

Receive L-PDUs are provided as constant structures statically generated from the communication matrix. They are arranged according to the corresponding hardware acceptance filter, so that there is one single list of receive CanIds for every Hardware Receive Object (HRH). The corresponding list can be derived by the HRH, if multiple *BasicCAN* objects are used. The subsequent filtering is the search through one list of multiple CanIds by comparing them with the new received CanId. In case of a hit the Receive L-PDU is derived from the found CanId.

[SWS_CANIF_00030] ⌈If the CanId of the received L-PDU in the HRH is configured to be received, then CanIf shall accept this L-PDU and the software filtering algorithm shall derive the corresponding Receive L-PDU from the found CanId.⌋

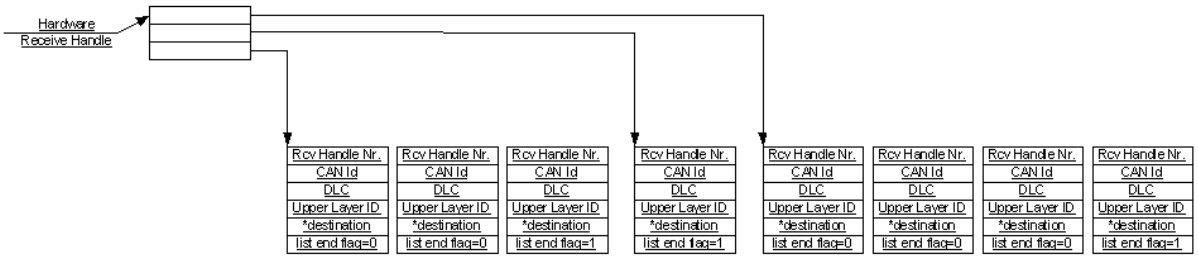


Figure 7.9: Software filtering example

[SWS_CANIF_00852] ⌈If a range is (partly) contained in another range, or a single CanId is contained in a range, the software filter shall select the L-PDU based on the following assumptions:

- A single CanId is always more relevant than a range.
- A smaller range is more relevant than a larger range.

⌋

7.20.2 Software filter algorithms

The choice of suitable software search algorithms it is up to the implementation of CanIf. According to the wide range of possible receive *BasicCAN* operations provided by the CAN Controller it is recommended to offer several search algorithms like linear search, table search and/or hash search variants to provide the most optimal solution for most use cases.

7.21 Data Length Check

The received Data Length value is compared with the configured Data Length value of the received L-PDU. The configured Data Length value shall be derived from the size of used bytes inside this L-PDU. The configured Data Length value may not be necessarily that Data Length value defined in the CAN communication matrix and used by the sender of this CAN L-PDU.

[SWS_CANIF_00026] ⌈CanIf shall accept all received L-PDUs (see [SWS_CANIF_00390]) with a Data Length value equal or greater than the configured Data Length value (see CanIfRxPduDataLength).⌋

[SWS_CANIF_00902] ⌈The Data Length Check shall be processed if it is enabled globally (see CanIfPrivateDataLengthCheck) and not disabled individually per PDU (see CanIfRxPduDataLengthCheck).⌋

Hint: If the Data Length Check is disabled globally, it can't be enabled individually per PDU.

[SWS_CANIF_00168] ⌈If the Data Length Check rejects a received L-PDU (see [SWS_CANIF_00026]), CanIf shall report runtime error code CANIF_E_INVALID_DATA_LENGTH to the Det_ReportRuntimeError() service of the DET module.⌋

[SWS_CANIF_00829] ⌈CanIf shall pass the received (see [SWS_CANIF_00006]) length value to LSduR, if the Data Length Check is passed.⌋

[SWS_CANIF_00830] ⌈CanIf shall pass the received (see [SWS_CANIF_00006]) length value to the LSduR, if the Data Length Check is not configured (see CanIf-PrivateDataLengthCheck and CanIfRxPduDataLengthCheck)⌋

7.22 Polling mode

The polling mode provides handling of transmit, receive and error events occurred in the CAN hardware without the usage of hardware interrupts. Thus the CanIf and the CanDrv provides notification services for detection and execution corresponding hardware events. In polling mode the behavior of these CanIf notification services does not change. By this way upper layer modules are abstracted from the strategy to detect hardware events. If different CanDrvs are in use, the calling frequency has to be harmonized during configuration setup and system integration.

These notification services are able to detect new events that occurred in the CAN hardware objects since its last execution. The CanIf's notification services for forwarding of detected events by the CanDrv are the same like for interrupt operation (see Chapter 8.3 "Function definitions").

The user has to consider, that the CanIf has to be able to perform notification services triggered by interrupt on interrupt level as well as to perform invoked notification services on task level. If any access to the CAN controller's mailbox is blocked, subsequent transmit buffering takes place (refer Chapter 7.11 "Transmit buffering").

The Polling and Interrupt mode can be configured for each underlying CAN controller.

7.23 Multiple CAN Driver support

CanIf needs a specific mapping to cover multiple CanDrv to provide a common interface to upper layers. Thus, CanIf must dispatch all actions up-down to the APIs of the corresponding CanDrv and underlying CAN Controller(s). For the way down-up CanIf has to provide adequate callback notifications to differentiate between multiple CanDrvs.

Each CanDrv supports a certain number of underlying CAN Controllers and a fixed number of HTHs/HRHs. Each CanDrv has an own numbering area, which starts always at zero for CAN Controllers and HTHs. CanIf has to derive the corresponding CanDrv from the L-SDU passed in the APIs. The parameters have to be translated accordingly: i.e. L-SDU => HTH/HRH, CanId, Data Length."

The support for multiple CanDrvs can be enabled and disabled by the configuration parameter CanIfPublicMultipleDrvSupport.

7.23.1 Transmit requests by using multiple CAN Drivers

Each Transmit L-PDU enables CanIf to derive the corresponding CAN Controller and implicitly CanDrv serving the affected Hardware Unit. Resolving of these dependencies is possible because of the construction of the *CAN Controller Handle*: it combines *CanDrv Handle* and the corresponding CAN Controller in the Hardware Unit.

At configuration time a CAN Controller Handle will be mapped to each CAN Controller. The sequence diagram Figure 7.10 below demonstrates two transmit requests directed to different CanDrvs. CanIf needs only to select the corresponding CanDrv in order to call the correct API service.

Note: Figure 7.10 and the following table serve only as an example. Finally, it is up to the implementation to access the correct APIs of underlying CanDrvs.

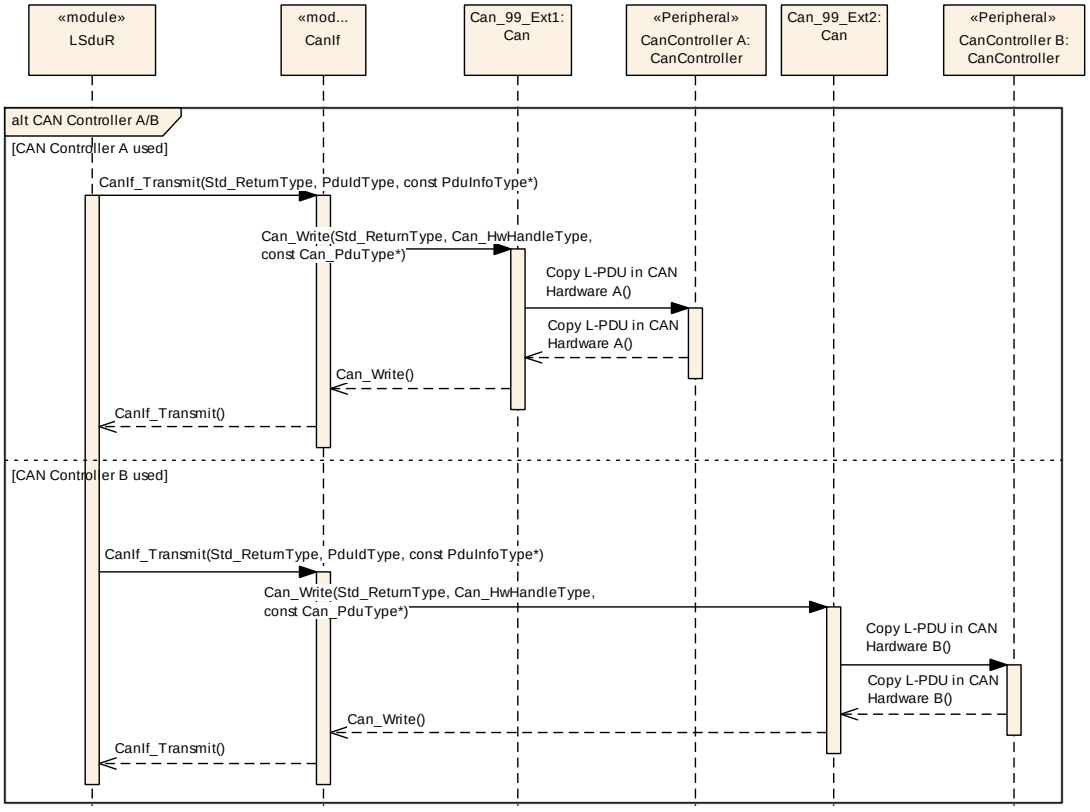


Figure 7.10: Transmission request with multiple CAN Drivers - simplified

Operations called	Description
CanIf_Transmit(PduId_1, PduInfoPtr_1)	Upper layer initiates a *transmit request*. The PduId is used for tracing the requested CAN Controller and then to serving the Hardware Unit. The number of the Hardware Unit is relevant for the dispatch as it is used as index for the array with pointer to functions. At first the number of the PDU channel group will be extracted from the PduId_1. Each PDU channel group refers to a CAN channel and thus as well the *Hardware Unit Number* and the *CAN Controller Number*. The *Hardware Unit Number* points on an instance of CanDrv and therefore refers all API services configured for the used Hardware Unit(s). One of these services is the requested transmit service.
Can_Write(Hth, PduInfoPtr)	Request for transmission to the corresponding CAN_Driver serving i.e. CAN Controller #0 within the "A" Hardware Unit.

Hardware request	All L-PDU data will be set in the Hardware of i.e. CAN Controller #0 within Hardware Unit "A" and the transmit request enabled.
CanIf_Transmit(PduId_2, PduInfoPtr_2)	Upper layer initiates Transmit Request. The PduId leads to another CAN Controller and then to another Hardware Unit. The number of the Hardware Unit is relevant for the dispatch as it is used as index for the array with pointer to functions. At first the number of the PDU channel group will be extracted from the PduId_2. Each PDU channel group refers to a CAN channel and thus as well to the *Hardware Unit Number* and to the *CAN Controller Number*. The *Hardware Unit Number* points on an instance of CanDrv and therefore refers all API services configured for the used Hardware Unit(s). One of these services is the requested transmit service.
Can_Write(Hth, PduInfoPtr_2)	Request for transmission to the corresponding CAN_Driver serving i.e. CAN Controller #1 within the "B" Hardware Unit.
Hardware request	All L-PDU data will be set in the Hardware of i.e. CAN Controller #1 within Hardware Unit "B" and the transmit request enabled.

7.23.2 Notification mechanism using multiple CAN Drivers

Even if multiple CanDrvs are used in a single ECU Every notification callback service invoked by CanDrvs at the CanIf exists only once. This means, that CanIf has to identify calling CanDrv using the passed parameters. CanIf identifies the calling CanDrv from the ControllerId within the Mailbox (Can_HwType) structure.

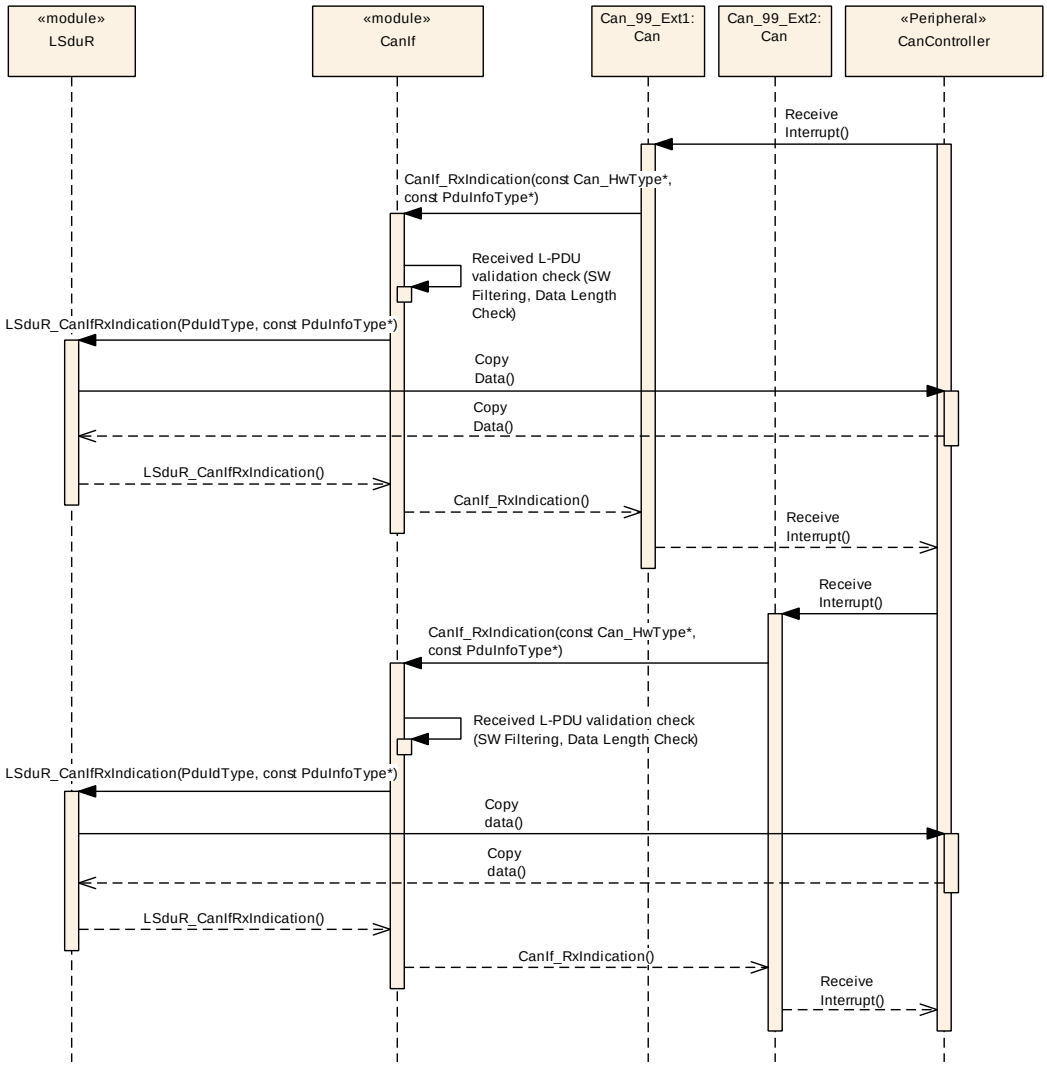


Figure 7.11: Receive interrupt with multiple CanDrvs - simplified

Operations called	Description
Receive Interrupt	CAN Controller 1 signals a successful reception and triggers a *receive interrupt*. The *ISR* of CanDrv A is invoked.
CanIf_RxIndication (Mailbox_1, PduInfoPtr_1)	The reception is indicated to CanIf by calling of CanIf_RxIndication(). The pointer Mailbox_1 identifies the HRH and its corresponding CAN Controller, which contains the received L-PDU specified by PduInfoPtr_1.
Validation check (SW Filtering, Data Length Check)	The Software Filtering checks, whether the Received L-PDU will be processed on a local ECU. If not, the Received L-SDU is not indicated to upper layers and further processing is suppressed. If the L-PDU is found, the Data Length of the Received L-PDU is compared with the expected, statically configured one for the received L-PDU.
LSduR_CanIfRxIndication(CanRxPduId_1, CanPduInfoPtr_1)	The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter CanRxPduId_1 specifies the ID of the received L-SDU. The second parameter is the reference on PduInfoType which provides access to the buffer containing the L-SDU.

Receive Interrupt	The CAN Controller 2 signals a successful reception and triggers a *receive interrupt*. The *ISR* of CanDrv B is invoked.
CanIf_RxIndication (Mailbox_2, PduInfoPtr_2)	The reception is indicated to CanIf by calling of CanIf_RxIndication(). The pointer Mailbox_2 identifies the HRH and its corresponding CAN Controller, which contains the received L-PDU specified by PduInfoPtr_2.
Validation check (SW Filtering, Data Length Check)	The Software Filtering checks, whether the Received L-PDU will be processed on a local ECU. If not, the Received L-SDU is not indicated to upper layers and further processing is suppressed. If the L-PDU is found, the Data Length of the Received L-PDU is compared with the expected, statically configured one for the received L-PDU.
LSduR_CanIfRxIndication(CanRxPduId_2, CanPduInfoPtr_2)	The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter CanRxPduId_2 specifies the ID of the received L-SDU. The second parameter is the reference on PduInfoType which provides access to the buffer containing the L-SDU.

7.24 Partial Networking

[SWS_CANIF_00747] ⌈If *Partial Networking* (PN) is enabled (see CanIfPublicPnSupport), CanIf shall support a PnTxFilter per CAN Controller which overlays the *PDU channel modes*.⌋

[SWS_CANIF_00748] ⌈The PnTxFilter of [SWS_CANIF_00747] shall only have an effect and transition its modes (enabled/disabled) if more than zero Tx L-PDUs per CAN Controller are configured as CanIfTxPduPnFilterPdu.⌋

[SWS_CANIF_00863] ⌈PnTxFilter shall be enabled during initialization (ref. to [SWS_CANIF_00747] and [SWS_CANIF_00748]).⌋

[SWS_CANIF_00749] ⌈If CanIf_SetControllerMode(ControllerId, CAN_CS_SLEEP) is called the PnTxFilter of the corresponding CAN Controller shall be enabled (ref. to [SWS_CANIF_00748] and [SWS_CANIF_00747]).⌋

[SWS_CANIF_00750] ⌈If the PnTxFilter of a CAN Controller is enabled, CanIf shall block all Tx requests to that CAN Controller (return E_NOT_OK when CanIf_Transmit() is called), except if the requested Tx L-PDUs is one of the configured CanIfTxPduPnFilterPdus of that CAN Controller. These CanIfTxPduPnFilterPdus shall always be passed to the corresponding CAN Driver.⌋

[SWS_CANIF_00751] ⌈If CanIf_TxConfirmation() is called, the corresponding PnTxFilter shall be disabled (ref. to [SWS_CANIF_00747] and [SWS_CANIF_00748]).⌋

[SWS_CANIF_00896] ⌈If CanIf_RxIndication() is called and PnTxFilter is enabled, the corresponding PnTxFilter shall be disabled (ref. to [SWS_CANIF_00747] and [SWS_CANIF_00748]).⌋

[SWS_CANIF_00752] ⌈If the PnTxFilter of a CAN Controller is disabled, CanIf shall behave as requested via CanIf_SetPduMode() (see [SWS_CANIF_00008]).⌋

[SWS_CANIF_00878] ⌈If CanIf_SetPduMode(ControllerId, CANIF_TX_OFFLINE) is called and Partial Networking is enabled (ref. to CanIfPublicPnSupport) the PnTxFilter of the corresponding CAN Controller shall be enabled (ref. to [SWS_CANIF_00748] and [SWS_CANIF_00747]).⌋

7.25 CAN FD Support

For performance reasons some CAN Controllers allow to use a Flexible Data-Rate feature called CAN FD (see [11, ISO 11898-1:2015]). Besides, the higher baud rate for the payload CAN FD also supports an extended payload which allows the transmission of up to 64 bytes. If these features are available depends on the general CAN FD support by the CAN Controller and if the CAN Controller is in CAN FD mode (valid CanControllerFdBaudrateConfig).

If an L-SDU shall be sent as CAN FD or conventional CAN 2.0 frame depends on the second most significant bit of the configured CanIfTxPduCanId. CanIf indicates this to CanDrv utilizing the same bit of PduInfo->id (Can_IdType) passed while calling Can_Write().

Note: If CanDrv is not in CAN FD mode (no CanControllerFdBaudrateConfig), the L-PDU will be sent as conventional CAN 2.0 frame as long as the SduLength <= 8 bytes.

Note: The arbitration phase of conventional CAN 2.0 frames and CAN FD frames does not differ if the same CanId is used. Therefore, even when using CAN FD frames each CanId must not be used more than once.

Which kind of frame was received by CanDrv is also indicated utilizing the second most significant bit of the Can_IdType passed with CanIf_RxIndication() (Mailbox->CanId). Based on this information CanIf decides how to map to the configured L-SDU (CanIfRxPduCfg) as described in [SWS_CANIF_00877].

Note: If upper layers don't care if a message was received by conventional CAN 2.0 frame or CAN FD frame, it is possible to use only one CanIfRxPduCfg for both types (using the second most significant bit of CanIfRxPduCanIdMask). This might allow local optimization. However, from a system point of view, the format for each frame has to be configured. Otherwise the sender wouldn't know which kind of frame shall be transmitted.

7.26 CAN XL Support

CAN XL (see [13, CiA610-1]) is an evolution from previous existing CAN 2.0 and CAN FD. It is more than just increased baudrate up to 20 MBit/s and increased payload up to 2048 bytes.

AUTOSAR specification mostly references to [13, CiA610-1] and [14, CiA611-1]. For overall picture, further specifications of following CiA series shall be considered:

- Series CiA 610: CAN XL specifications and test plans
- Series CiA 611: CAN XL higher-layer services
- Series CiA 612: CAN XL guidelines and application notes
- Series CiA 613: CAN XL add-on services

CAN XL introduces game changing attributes new to CAN:

- The Priority ID allows fully priority based bus access. In contrast to the previously used CAN ID of CAN 2.0 and CAN FD it adds separation of concerns, because the priority of a frame is no longer linked to the identification of the content. There is no further information encoded anymore other than priority.
- The SDU Type indicates the type of the protocol embedded in the data field; this allows e.g. transmission of content based CAN XL messages on the same bus as (tunneled) CAN 2.0 and CAN FD as well as tunneled Ethernet frames. Supported types are defined in [14, CiA 611-1].
- The Virtual CAN Network ID allows to separate the traffic on bus into virtual networks. This allows e.g. enhanced separation of concerns and grouping of ECUs. Furthermore it enables even low powered ECUs to early filter out received messages of a VCID it is not participating.
- The Acceptance Field interpretation is dependent of used SDU Type. E.g. for content based adressing this resembles the Message ID which is comparable to the CAN ID of CAN 2.0 and CAN FD. It may be also used for improved ingress filtering capabilites.
- *Transceiver mode switching* is used to allow baudrates above 8 Mbit/s. This switches the transceiver to use PWM coding in the XL Data phase, which allows bitrates up to 20 MBit/s. Please note, that in this case the interoperability with legacy CAN 2.0/CAN FD nodes on same bus is not possible. Furthermore, error signaling must currently be turned off.

The CAN XL feature could be used if the CAN Controller supports CAN XL and the CAN XL mode is enabled (existence of a CanXLBaudrateConfig).

Constraints:

- All other nodes on the same bus are expected to at least support CAN FD.

- Due to different handling of possibly large CAN XL buffers with quite diverse range of sizes, L-PDU transmission buffering within CanIf (see Chapter 7.11) is not available for CAN XL frames.

7.26.1 Dynamic Control of CAN XL Attributes

Instead of configuring all relevant CAN XL parameters for a CAN XL frame, some parameters may also be passed as MetaData:

During reception, only those parameters are matched, that are configured for the corresponding CanIfRxPduCfg, while the not configured parameters shall be passed on to the upper layer as MetaData.

During transmission, the actually used parameters are assembled from the passed MetaData items and the parameters configured for the CanIfTxPduCfg.

[SWS_CANIF_00937] ⌈When a CAN XL frame is received, the CAN XL parameters configured as MetaData items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32 for the global PDU shall be extracted from the passed CanXL_Params and collected into the MetaData of the PDU that is passed to the upper layer.⌋

[SWS_CANIF_00938] ⌈When a CAN XL frame shall be transmitted, the CanXL_Params shall be assembled from the CAN XL parameters passed as MetaData items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32 from the upper layer, and the configured CanIfTxPduXLParams.⌋

Please note: The configuration has to ensure that each CAN XL parameter of a received or transmitted CAN XL frame is either configured or passed as MetaData, never both. See also Chapter 10.2.

7.27 CANsec Support

CANsec (see [15, CiA 613-2]) is a secure communication protocol on OSI Layer 2. It internally reuses MACsec (see [16, IEEE-802.1AE-2018], [17, IEEE-802.1X-2020]) and can thus be split into two major components:

- **CANsec Security entity (CANsec SecY):** The CANsec SecY implements the secure communication on the network. It is typically implemented in the CAN Controller and can be accessed via the AUTOSAR CanDrv.
- **CANsec Key agreement entity (CANsec KaY):** The CANsec KaY communicates with other CANsec participants on the network to agree on the CANsec configuration, including keys. The CANsec KaY is currently not part of the AUTOSAR specification, but is planned to be implemented in the MKA. See also Section 4.1.

In the AUTOSAR architecture, the CanIf is located between the CANsec SecY and the CANsec KaY and relays the CANsec management API calls to the respective CanDrv. For usage of the respective APIs by the CANsec KaY, please refer to the MACsec sequence diagrams in [18, SWS MACsec Key Agreement] Chapter 9 *"Sequence diagrams"*.

[SWS_CANIF_00970] Forwarding CANsec-related calls to the CanDrv ⌈CanIf shall forward the following API calls to the CanDrv:

- a call of CanIf_CanSecUpdateSecY shall be forwarded to Can_CanSecUpdateSecY
- a call of CanIf_CanSecAddRxSc shall be forwarded to Can_CanSecAddRxSc
- a call of CanIf_CanSecDeleteRxSc shall be forwarded to Can_CanSecDeleteRxSc
- a call of CanIf_CanSecAddTxSa shall be forwarded to Can_CanSecAddTxSa
- a call of CanIf_CanSecUpdateTxSa shall be forwarded to Can_CanSecUpdateTxSa
- a call of CanIf_CanSecDeleteTxSa shall be forwarded to Can_CanSecDeleteTxSa
- a call of CanIf_CanSecAddRxSa shall be forwarded to Can_CanSecAddRxSa
- a call of CanIf_CanSecUpdateRxSa shall be forwarded to Can_CanSecUpdateRxSa
- a call of CanIf_CanSecDeleteRxSa shall be forwarded to Can_CanSecDeleteRxSa
- a call of CanIf_CanSecGetTxSaNextPn shall be forwarded to Can_CanSecGetTxSaNextPn
- a call of CanIf_CanSecGetCanSecSecYStatistics shall be forwarded to Can_CanSecGetCanSecSecYStatistics
- a call of CanIf_CanSecGetCanSecTxScStatistics shall be forwarded to Can_CanSecGetCanSecTxScStatistics
- a call of CanIf_CanSecGetCanSecRxScStatistics shall be forwarded to Can_CanSecGetCanSecRxScStatistics
- a call of CanIf_CanSecSetControlledPortEnabled shall be forwarded to Can_CanSecSetControlledPortEnabled

⌋

[SWS_CANIF_00971] Forwarding of CANsec related callback APIs calls towards MKA ⌈The CanIf shall forward the following API calls from the CanDrv to MKA:

- a call of CanIf_CanSecUpdateSecYNotification shall be forwarded to Mka_CanSecUpdateSecYNotification
- a call of CanIf_CanSecAddTxSaNotification shall be forwarded to Mka_CanSecAddTxSaNotification
- a call of CanIf_CanSecAddRxSaNotification shall be forwarded to Mka_CanSecAddRxSaNotification
- a call of CanIf_CanSecGetCanSecSecYStatisticsNotification shall be forwarded to Mka_GetCanSecSecYStatisticsNotification
- a call of CanIf_CanSecGetCanSecTxScStatisticsNotification shall be forwarded to Mka_GetCanSecTxScStatisticsNotification
- a call of CanIf_CanSecGetCanSecRxScStatisticsNotification shall be forwarded to Mka_GetCanSecRxScStatisticsNotification

⌋

7.27.1 Communication via controlled and uncontrolled port

When a CANsec-protected CAN network is being started, secure communication can only happen once CANsec-secured communication is established, i.e., the controlled port is set to enabled by the CANsec KaY. Unsecured communication over the uncontrolled port (specified by the bypass rules in the CAN Driver) can happen earlier, i.e., as soon as the CAN Controller indicated CAN_CS_STARTED. To this end, CanIf indicates mode changes of CAN controllers to the CANsec KaY via Mka_CanControllerModeIndication (see [SWS_CANIF_00711]).

The uncontrolled and controlled port (or multiple controlled ports if more than one SecY is configured) can be represented in AUTOSAR by means of routing path groups, which are switched by the BswM. To this end, the CANsec KaY needs to call BswM_Mka_ControlledPortStatus to indicate that the controlled port for a specific SecY has been activated and that CANsec-secured communication is now possible. The BswM will then switch the routing path group representing the controlled port to active, thus enabling secured communication.

7.28 Security Events

[SWS_CANIF_91010] Security events for CanIf ⌈

Name	Description	ID
SEV_CAN_TX_ERROR_DETECTED	A transmission related error was detected. Depending on the context data this could indicate suspicious CAN activity.	19
SEV_CAN_RX_ERROR_DETECTED	A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity.	20
SEV_CAN_ERRORSTATE_PASSIVE	The CAN controller transitioned to state passive.	21
SEV_CAN_ERRORSTATE_BUSOFF	The CAN controller transitioned to state busoff.	22

⌋

[SWS_CANIF_00913] ⌈If security event reporting has been enabled for the CanIf module (CanIfEnableSecurityEventReporting = true) the respective security events shall be reported to the IdsM via the interfaces defined in [8, AUTOSAR_CP_SWS_BSWGeneral].⌋

[SWS_CANIF_00915] ⌈If CanIf_ErrorNotification() is called by CanDrv, the function shall evaluate whether a Tx related error was detected. If this is the case the CanIfshall report the security event SEV_CAN_TX_ERROR_DETECTED.⌋

[SWS_CANIF_92000] Security event context data definition: SEV_CAN_TX_ERROR_DETECTED ⌈

SEV Name	SEV_CAN_TX_ERROR_DETECTED	
ID	19	
Description	A transmission related error was detected. Depending on the context data this could indicate suspicious CAN activity.	
Context Data Version	1	
Context Data	**Data Type**	**Allowed Values**
ControllerId	uint8	
CanError	uint8	CAN_ERROR_BIT_MONITORING1 (0x01) CAN_ERROR_BIT_MONITORING0 (0x02) CAN_ERROR_BIT (0x03) CAN_ERROR_CHECK_ACK_FAILED (0x04) CAN_ERROR_ACK_DELIMITER (0x05) CAN_ERROR_ARBITRATION_LOST (0x06) CAN_ERROR_OVERLOAD (0x07)

⌋

[SWS_CANIF_00916] ⌈If CanIf_ErrorNotification() is called by CanDrv, the function shall evaluate whether a Rx related error was detected. If this is the case the CanIf shall report the security event SEV_CAN_RX_ERROR_DETECTED.⌋

[SWS_CANIF_92001] Security event context data definition: SEV_CAN_RX_ERROR_DETECTED ⌈

SEV Name	*SEV_CAN_RX_ERROR_DETECTED*	
ID	20	
Description	A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity.	
Context Data Version	1	
Context Data	*Data Type*	*Allowed Values*
ControllerId	uint8	
CanError	uint8	CAN_ERROR_CHECK_FORM_FAILED (0x08) CAN_ERROR_CHECK_STUFFING_FAILED (0x09) CAN_ERROR_CHECK_CRC_FAILED (0x0A) CAN_ERROR_BUS_LOOK (0x0B)

⌋

[SWS_CANIF_00917] ⌈If `CanIf_ControllerErrorStatePassive()` is called by `CanDrv`, the `CanIf` shall report the security event `SEV_CAN_ERRORSTATE_PASSIVE` in following cases:

- TxErrorCounter > 127 and TxErrorCounter <= 255
- RxErrorCounter > 127 and TxErrorCounter <= 255

⌋

[SWS_CANIF_92002] Security event context data definition: SEV_CAN_ERRORSTATE_PASSIVE ⌈

SEV Name	*SEV_CAN_ERRORSTATE_PASSIVE*	
ID	21	
Description	The CAN controller transitioned to state passive.	
Context Data Version	1	
Context Data	*Data Type*	*Allowed Values*
ControllerId	uint8	
ErrorCounterThreshold	uint8	0x00: TxErrorCounter > 127 AND RxErrorCounter > 127 0x01: TxErrorCounter > 127 AND RxErrorCounter < 127 0x02: TxErrorCounter < 127 AND RxErrorCounter > 127

⌋

[SWS_CANIF_00918] ⌈If CanIf_ControllerBusOff is called by `CanDrv`, the `CanIf` shall report the security event `SEV_CAN_ERRORSTATE_BUSOFF`.⌋

[SWS_CANIF_92003] Security event context data definition: SEV_CAN_ERRORSTATE_BUSOFF ⌈

SEV Name	*SEV_CAN_ERRORSTATE_BUSOFF*	
ID	22	
Description	The CAN controller transitioned to state busoff.	
Context Data Version	1	
Context Data	*Data Type*	*Allowed Values*

▽

△

SEV Name	*SEV_CAN_ERRORSTATE_BUSOFF*	
ControllerId	uint8	

⌋

7.29 Error classification

7.29.1 Development Errors

[SWS_CANIF_91006] Definition of development errors in module CanIf ⌈

Type of error	*Related error code*	*Error value*
API service called with invalid CAN ID	CANIF_E_PARAM_CANID	10
API service called with invalid hardware object	CANIF_E_PARAM_HOH	12
API service called with invalid PDU ID	CANIF_E_PARAM_LPDU	13
API service called with invalid controller ID	CANIF_E_PARAM_CONTROLLERID	15
API service called with invalid wakeup source	CANIF_E_PARAM_WAKEUPSOURCE	16
API service called with invalid transceiver ID	CANIF_E_PARAM_TRCV	17
API service called with invalid transceiver mode	CANIF_E_PARAM_TRCVMODE	18
API service called with invalid transceiver wakeup mode	CANIF_E_PARAM_TRCVWAKEUPMODE	19
API service called with invalid pointer	CANIF_E_PARAM_POINTER	20
API service called with invalid controller mode	CANIF_E_PARAM_CTRLMODE	21
API service called with invalid PDU mode	CANIF_E_PARAM_PDU_MODE	22
API services called with invalid parameter	CANIF_E_PARAM_CAN_ERROR	23
API service used without module initialization	CANIF_E_UNINIT	30
Transmit PDU ID invalid	CANIF_E_INVALID_TXPDUID	50
Receive PDU ID invalid	CANIF_E_INVALID_RXPDUID	60
CAN Interface initialisation failed	CANIF_E_INIT_FAILED	80

⌋

7.29.2 Runtime Errors

[SWS_CANIF_91007] Definition of runtime errors in module CanIf ⌈

Type of error	*Related error code*	*Error value*
Failed Data Length Check	CANIF_E_INVALID_DATA_LENGTH	61
Data Length	CANIF_E_DATA_LENGTH_MISMATCH	62
Transmit requested on offline PDU channel	CANIF_E_STOPPED	70
Message length was exceeding the maximum length	CANIF_E_TXPDU_LENGTH_EXCEEDED	90

⌋

7.29.3 Production Errors

There are no production errors.

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed.

[SWS_CANIF_00142] Definition of imported datatypes of module CanIf

Upstream requirements: SRS_BSW_00348, SRS_BSW_00353

⌈

Module	*Header File*	*Imported Type*
Can	Can_GeneralTypes.h	Can_ControllerStateType
	Can_GeneralTypes.h	Can_ErrorStateType
	Can_GeneralTypes.h	Can_ErrorType
	Can_GeneralTypes.h	Can_HwHandleType
	Can_GeneralTypes.h	Can_HwType
	Can_GeneralTypes.h	Can_IdType
	Can_GeneralTypes.h	Can_PduType
	Can_GeneralTypes.h	Can_TimeStampType (draft)
CanTrcv	Can_GeneralTypes.h	CanTrcv_TrcvModeType
	Can_GeneralTypes.h	CanTrcv_TrcvWakeupModeType
	Can_GeneralTypes.h	CanTrcv_TrcvWakeupReasonType
CanXL	Can_GeneralTypes.h	CanXL_HwType
	Can_GeneralTypes.h	CanXL_Params
	Can_GeneralTypes.h	CanXL_PduType
Comtype	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
EcuM	EcuM.h	EcuM_WakeupSourceType
IdsM	IdsM.h	IdsM_SecurityEventIdType
	Rte_IdsM_Type.h	IdsM_TimestampDataType
Mka	Mka.h	Mka_CanSecConfigType
	Mka.h	Mka_ConfidentialityOffsetType (draft)
	Mka.h	Mka_SakKeyPtrType (draft)
	Mka.h	Mka_Stats_Rx_ScType (draft)
	Mka.h	Mka_Stats_Rx_SecYType (draft)
	Mka.h	Mka_Stats_Tx_ScType (draft)
	Mka.h	Mka_Stats_Tx_SecYType (draft)
	Mka.h	Mka_ValidateFramesType (draft)
Std	Std_Types.h	Std_ReturnType

⌋

8.2 Type definitions

8.2.1 CanIf_ConfigType

[SWS_CANIF_00144] Definition of datatype CanIf_ConfigType ⌈

Name	CanIf_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	The contents of the initialization data structure are CAN interface specific
Description	This type defines a data structure for the post build parameters of the CAN interface for all underlying CAN drivers. At initialization the CanIf gets a pointer to a structure of this type to get access to its configuration data, which is necessary for initialization.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00523] ⌈The initialization data structure for a specific `CanIf_ConfigType` shall include the definition of `CanIf` public parameters and the definition for each `L-PDU`/`L-SDU`.⌋

Note: The definition of `CanIf` public parameters and the definition for each `L-PDU`/`L-SDU` are specified in Chapter 10.

8.2.2 CanIf_PduModeType

[SWS_CANIF_00137] Definition of datatype CanIf_PduModeType ⌈

Name	CanIf_PduModeType		
Kind	Enumeration		
Range	CANIF_OFFLINE	0x00	= 0 Transmit and receive path of the corresponding channel are disabled => no communication mode
	CANIF_TX_OFFLINE	0x01	Transmit path of the corresponding channel is disabled. The receive path is enabled.
	CANIF_TX_OFFLINE_ ACTIVE	0x02	Transmit path of the corresponding channel is in offline active mode (see [SWS_CANIF_00072]). The receive path is enabled. This mode requires CanIfTxOffline ActiveSupport = TRUE.
	CANIF_ONLINE	0x03	Transmit and receive path of the corresponding channel are enabled => full operation mode
Description	The PduMode of a channel defines its transmit or receive activity. Communication direction (transmission and/or reception) of the channel can be controlled separately or together by upper layers.		
Available via	CanIf.h		

⌋

8.2.3 CanIf_NotifStatusType

[SWS_CANIF_00201] Definition of datatype CanIf_NotifStatusType ⌈

Name	CanIf_NotifStatusType		
Kind	Enumeration		
Range	CANIF_TX_RX_ NOTIFICATION	–	The requested Rx/Tx CAN L-PDU was successfully transmitted or received.
	CANIF_NO_NOTIFICATION	0x00	No transmit or receive event occurred for the requested L-PDU.
Description	Return value of CAN L-PDU notification status.		
Available via	CanIf.h		

⌋

8.3 Function definitions

This chapter contains all APIs provided by the module.

[SWS_CANIF_00661] ⌈All CanIf API services other than `CanIf_Init()` shall not execute their normal operation and return `E_NOT_OK` unless the `CanIf` has been initialized with a preceding call of `CanIf_Init()`.⌋

8.3.1 CanIf_Init

[SWS_CANIF_00001] Definition of API function CanIf_Init

Upstream requirements: SRS_BSW_00405, SRS_BSW_00101, SRS_BSW_00358, SRS_BSW_00414

⌈

Service Name	CanIf_Init	
Syntax	`void CanIf_Init ( const CanIf_ConfigType* ConfigPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration parameter set, used e.g. for post build parameters
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service Initializes internal and external interfaces of the CAN Interface for the further processing.	
Available via	CanIf.h	

⌋

Note: All underlying CAN controllers and transceivers still remain not operational.

Note: The service `CanIf_Init()` is called only by the `EcuM`.

[SWS_CANIF_00085] ⌈The service `CanIf_Init()` shall initialize the global variables and data structures of the `CanIf` including flags and buffers.⌋

8.3.2 CanIf_DeInit

[SWS_CANIF_91002] Definition of API function CanIf_DeInit

Upstream requirements: SRS_BSW_00336

⌈

Service Name	CanIf_DeInit

Syntax	`void CanIf_DeInit (`
	`  void`
	`)`

Service ID [hex]	0x02
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	De-initializes the CanIf module.
Available via	CanIf.h

⌋

Note: General behavior and constraints on de-initialization functions are specified by *[SWS_BSW_00152]*, *[SWS_BSW_00072]*, *[SWS_BSW_00232]*, *[SWS_BSW_00233]*.

Caveat: Caller of the `CanIf_DeInit()` function has to be sure there are no on-going transmissions/receptions, nor any pending transmission confirmations.

8.3.3 CanIf_SetControllerMode

[SWS_CANIF_00003] Definition of API function CanIf_SetControllerMode ⌈

Service Name	CanIf_SetControllerMode	

Syntax	`Std_ReturnType CanIf_SetControllerMode (`
	`  uint8 ControllerId,`
	`  Can_ControllerStateType ControllerMode`
	`)`	

Service ID [hex]	0x03	
Sync/Async	Asynchronous	
Reentrancy	Reentrant (Not for the same controller)	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for mode transition.

▽

△

	ControllerMode	Requested mode transition
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Controller mode request has been accepted E_NOT_OK: Controller mode request has not been accepted
Description	This service calls the corresponding CAN Driver service for changing of the CAN controller mode.	
Available via	CanIf.h	

⌋

Note: The service CanIf_SetControllerMode() initiates a transition to the requested CAN controller mode ControllerMode of the CAN controller which is assigned by parameter ControllerId.

[SWS_CANIF_00308] ⌈The service CanIf_SetControllerMode() shall call Can_SetControllerMode(Controller, Transition) for the requested CAN controller.⌋

[SWS_CANIF_00311]

Upstream requirements: SRS_BSW_00323

⌈If parameter ControllerId of CanIf_SetControllerMode() has an invalid value, the CanIf shall report development error code CANIF_E_PARAM_CONTROLLERID to the Det_ReportError service of the DET module, when CanIf_SetControllerMode() is called.⌋

[SWS_CANIF_00774]

Upstream requirements: SRS_BSW_00323

⌈If parameter ControllerMode of CanIf_SetControllerMode() has an invalid value (not CAN_CS_STARTED, CAN_CS_SLEEP or CAN_CS_STOPPED), the CanIfshall report development error code CANIF_E_PARAM_CTRLMODE to the Det_ReportError service of the DET module, when CanIf_SetControllerMode() is called.⌋

Note: The ID of the CAN controller is published inside the configuration description of the CanIf.

8.3.4 CanIf_GetControllerMode

[SWS_CANIF_00229] Definition of API function CanIf_GetControllerMode ⌈

Service Name	CanIf_GetControllerMode

Syntax	`Std_ReturnType CanIf_GetControllerMode (`
	`  uint8 ControllerId,`
	`  Can_ControllerStateType* ControllerModePtr`
	`)`

▽

△

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for current operation mode.
Parameters (inout)	None	
Parameters (out)	ControllerModePtr	Pointer to a memory location, where the current mode of the CAN controller will be stored.
Return value	Std_ReturnType	`E_OK`: Controller mode request has been accepted. `E_NOT_OK`: Controller mode request has not been accepted.
Description	This service calls the corresponding CAN Driver service for obtaining the current status of the CAN controller.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00313]

Upstream requirements: SRS_BSW_00323

⌈If parameter `ControllerId` of `CanIf_GetControllerMode()` has an invalid, the CanIf shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET, when `CanIf_GetControllerMode()` is called.⌋

[SWS_CANIF_00656]

Upstream requirements: SRS_BSW_00323

⌈If parameter `ControllerModePtr` of `CanIf_GetControllerMode()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_POINTER` to the `Det_ReportError` service of the DET, when `CanIf_GetControllerMode()` is called.⌋

Note: The ID of the CAN controller module is published inside the configuration description of the CanIf.

8.3.5 CanIf_GetControllerErrorState

[SWS_CANIF_91001] Definition of API function CanIf_GetControllerErrorState ⌈

Service Name	CanIf_GetControllerErrorState

Syntax	`Std_ReturnType CanIf_GetControllerErrorState (`
	`  uint8 ControllerId,`
	`  Can_ErrorStateType* ErrorStatePtr`
	`)`

Service ID [hex]	0x4b
Sync/Async	Synchronous
Reentrancy	Non Reentrant for the same ControllerId

▽

△

Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState.
Parameters (inout)	None	
Parameters (out)	ErrorStatePtr	Pointer to a memory location, where the error state of the CAN controller will be stored.
Return value	Std_ReturnType	E_OK: Error state request has been accepted. E_NOT_OK: Error state request has not been accepted.
Description	This service calls the corresponding CAN Driver service for obtaining the error state of the CAN controller.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00898]

Upstream requirements: SRS_BSW_00323

⌈If parameter ControllerId of CanIf_GetControllerErrorState() has an invalid value, the CanIf shall report development error code CANIF_E_PARAM_CONTROLLERID to the Det_ReportError service of the DET, when CanIf_GetControllerErrorState() is called.⌋

[SWS_CANIF_00899]

Upstream requirements: SRS_BSW_00323

⌈If parameter ErrorStatePtr of CanIf_GetControllerErrorState() is a null pointer, the CanIf shall report development error code CANIF_E_PARAM_POINTER to the Det_ReportError service of the DET, when CanIf_GetControllerErrorState() is called.⌋

8.3.6 CanIf_Transmit

[SWS_CANIF_00005] Definition of API function CanIf_Transmit ⌈

Service Name	CanIf_Transmit	
Syntax	`Std_ReturnType CanIf_Transmit ( PduIdType TxPduId, const PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x49	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	Identifier of the PDU to be transmitted
	PduInfoPtr	Length of and pointer to the PDU data and pointer to MetaData.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Transmit request has been accepted. E_NOT_OK: Transmit request has not been accepted.
Description	Requests transmission of a PDU.	

▽

△

Available via	CanIf.h

⌋

Note: The corresponding CAN Controller and HTH have to be resolved by the TxPduId.

[SWS_CANIF_00939] ⌈When CanIf_Transmit() is called and the container CanIfTxPduCfg.CanIfTxPduXLParams is existing which indicates a CAN XL frame, then CanXL_Write() shall be used for transmission. Otherwise Can_Write() shall be used for transmission.⌋

[SWS_CANIF_00317] ⌈The service CanIf_Transmit() shall not accept a transmit request, if the controller mode referenced by ControllerId is different to CAN_CS_STARTED and the channel mode at least for the transmit path is not online or offline active.⌋

[SWS_CANIF_00318] ⌈CanIf_Transmit() shall call Can_Write() with the hardware transmit handle corresponding to the provided TxPduId and a Can_PduType structure where:

- swPduHandle is set to the CanTxPduId used in the corresponding CanIf_TxConfirmation() call
- length is set to the value provided as PduInfoPtr->SduLength, possibly reduced according to [SWS_CANIF_00894]
- id is set to the CAN ID associated with the TxPduId
- sdu is set to the pointer provided as PduInfoPtr->SduDataPtr

⌋

Note: PduInfoPtr is a pointer to a L-SDU user memory, *CAN Identifier*, L-SDU handle and Data Length (see [1, Specification of CAN Driver]).

[SWS_CANIF_00957] ⌈CanIf_Transmit() shall call CanXL_Write() with the hardware transmit handle Hth corresponding to the provided TxPduId, AsyncBuffer set to the value of Pdu.KeepLocalPduBuffer of the global PDU, and a CanXL_PduType structure where:

- swPduHandle is set to the CanTxPduId used in the corresponding CanIf_TxConfirmation() call
- length is set to the value provided as PduInfoPtr->SduLength
- sdu is set to the pointer provided as PduInfoPtr->SduDataPtr
- XLParams is set to a pointer to a CanXL_Params structure containing the PriorityId, Vcid, SduType, AcceptanceField, and Sec (the Sec element is currently not supported and always set to 0)

⌋

[SWS_CANIF_00882] ⌈`CanIf_Transmit()` shall accept a `NULL` pointer as `PduInfoPtr->SduDataPtr`, if the PDU is configured for triggered transmission: `CanIfTxPduTriggerTransmit = TRUE`.⌋

[SWS_CANIF_00162] ⌈If the call of `Can_Write()`/`CanXL_Write()` returns `E_OK` the transmit request service `CanIf_Transmit()` shall return `E_OK`.⌋

Note: If the call of `Can_Write()`/`CanXL_Write()` returns `E_NOT_OK`, then the transmit request service `CanIf_Transmit()` shall return `E_NOT_OK`. If the transmit request service `CanIf_Transmit()` returns `E_NOT_OK`, then the upper layer module is responsible to repeat the transmit request.

[SWS_CANIF_00319]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TxPduId` of `CanIf_Transmit()` has an invalid value, `CanIf` shall report development error code `CANIF_E_INVALID_TXPDUID` to the `Det_ReportError` service of the DET, when `CanIf_Transmit()` is called.⌋

[SWS_CANIF_00320]

Upstream requirements: SRS_BSW_00323

⌈If parameter `PduInfoPtr` of `CanIf_Transmit()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_POINTER` to the `Det_ReportError` service of the DET module, when `CanIf_Transmit()` is called.⌋

[SWS_CANIF_00893] ⌈When `CanIf_Transmit()` is called with `PduInfoPtr->SduLength` exceeding the maximum length of the PDU referenced by `TxPduId`:

- `SduLength > 8` if the `Can_IdType` indicates a classic CAN frame
- `SduLength > 64` if the `Can_IdType` indicates a CAN FD frame
- `SduLength > 2048` if the container `CanIfTxPduCfg.CanIfTxPduXLParams` is existing which indicates a CAN XL frame

`CanIf` shall report runtime error code `CANIF_E_DATA_LENGTH_MISMATCH` to the `Det_ReportRuntimeError()` service of the *DET*.⌋

[SWS_CANIF_00940] ⌈When `CanIf_Transmit()` is called with `PduInfoPtr->SduLength` less than 1 and the container `CanIfTxPduCfg.CanIfTxPduXLParams` is existing which indicates a CAN XL frame, and `SDU Type` is configured to a value different from 03h (tunneled CAN 2.0/FD frame), `CanIf` shall report runtime error code `CANIF_E_DATA_LENGTH_MISMATCH` to the `Det_ReportRuntimeError()` service of the *DET*.⌋

Note: Besides static configured transmissions there are dynamic transmissions, too. Therefore, the valid data length is always passed by PduInfoPtr->SduLength. Furthermore, even the frame type might change via CanIf_SetDynamicTxId(). [SWS_CANIF_00893] ensures that not matching transmit requests can be detected via DET.

[SWS_CANIF_00894] ⌈When CanIf_Transmit() is called with PduInfoPtr->SduLength exceeding the length of the global PDU (see Pdu.PduLength) referenced by TxPduId and CanIfTxPduTruncation is enabled, CanIf shall transmit data up to the length of the global PDU (see Pdu.PduLength) and discard the rest.⌋

[SWS_CANIF_00900] ⌈When CanIf_Transmit() is called with PduInfoPtr->SduLength exceeding the length of the global PDU (see Pdu.PduLength) referenced by TxPduId and CanIfTxPduTruncation is disabled, CanIf shall report the runtime error CANIF_E_TXPDU_LENGTH_EXCEEDED and return E_NOT_OK without further actions.⌋

Note: During the call of CanIf_Transmit() the buffer of PduInfoPtr is controlled by CanIf and this buffer should not be accessed for read/write from another call context. After return of this call the ownership changes to the upper layer.

[SWS_CANIF_00941] ⌈When CanIf_Transmit is called and the container CanIfTxPduCfg.CanIfTxPduXLParams is existing which indicates a CAN XL frame, the CanXL_Params passed to CanXL_Write shall be composed as described in [SWS_CANIF_00938] of the CAN XL parameters configured in CanIfTxPduXLParams, and those passed as MetaData via the PduInfoPtr->MetaDataPtr.⌋

As mentioned in Chapter 7.26.1, each CAN XL Param is either configured or passed via MetaData.

[SWS_CANIF_00942] ⌈For a tunneled CAN 2.0/FD frame (SDU Type 03h) the CAN ID shall by transmitted in the Acceptance Field. The CAN ID is either configured with the parameter CanIfTxPduCanId or passed as MetaData.⌋

[SWS_CANIF_00943] ⌈When CanIf_Transmit is called and CanXL_Write is used for transmission the parameter PduInfo->Sec shall be set to 0.⌋

8.3.7 CanIf_ReleaseRxBuffer

[SWS_CanIf_91040] Definition of API function CanIf_ReleaseRxBuffer ⌈

Service Name	CanIf_ReleaseRxBuffer

Syntax	`void CanIf_ReleaseRxBuffer (`
	`  PduIdType RxPduId`
	`)`

Service ID [hex]	0x63

▽

△

Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId	
Parameters (in)	RxPduId	Identifier of the received PDU.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Indication from the upper layer to release the lower layer reception buffer.	
Available via	CanIf.h	

⌋

8.3.8 CanIf_ReadRxPduData

[SWS_CANIF_00194] Definition of API function CanIf_ReadRxPduData ⌈

Service Name	CanIf_ReadRxPduData	
Syntax	`Std_ReturnType CanIf_ReadRxPduData ( PduIdType CanIfRxSduId, PduInfoType* CanIfRxInfoPtr )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CanIfRxSduId	Receive L-SDU handle specifying the corresponding CAN L-SDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
Parameters (inout)	None	
Parameters (out)	CanIfRxInfoPtr	Contains the length (SduLength) of the received PDU, a pointer to a buffer (SduDataPtr) containing the PDU, and the MetaData related to this PDU.
Return value	Std_ReturnType	`E_OK`: Request for L-SDU data has been accepted `E_NOT_OK`: No valid data has been received
Description	This service provides the Data Length and the received data of the requested CanIfRxSduId to the calling upper layer.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00324] ⌈The function `CanIf_ReadRxPduData()` shall not accept a request and return `E_NOT_OK`, if the corresponding controller mode referenced by `ControllerId` is different to `CAN_CS_STARTED` and the channel mode is in the receive path online.⌋

[SWS_CANIF_00325]

Upstream requirements: SRS_BSW_00323

⌈If parameter `CanIfRxSduId` of `CanIf_ReadRxPduData()` has an invalid value, e.g. not configured to be stored within `CanIf` via `CanIfRxPduReadData`, `CanIf` shall report development error code `CANIF_E_INVALID_RXPDUID` to the `Det_ReportError` service of the DET, when `CanIf_ReadRxPduData()` is called.⌋

[SWS_CANIF_00326]

Upstream requirements: SRS_BSW_00323

⌈If parameter CanIfRxInfoPtr of CanIf_ReadRxPduData() has an invalid value, CanIf shall report development error code CANIF_E_PARAM_POINTER to the Det_ReportError service of the DET module, when CanIf_ReadRxPduData() is called.⌋

[SWS_CANIF_00329] ⌈CanIf_ReadRxPduData() shall not be used for CanIfRxSduId, which are defined to receive multiple CAN-Ids (range reception).⌋

Note: During the call of CanIf_ReadRxPduData() the buffer of CanIfRxInfoPtr is controlled by CanIf and this buffer should not be accessed for read/write from another call context. After return of this call the ownership changes to the upper layer.

[SWS_CANIF_00330] ⌈Configuration of CanIf_ReadRxPduData(): This API can be enabled or disabled at pre-compile time configuration by the configuration parameter CanIfPublicReadRxPduDataApi.⌋

8.3.9 CanIf_ReadTxNotifStatus

[SWS_CANIF_00202] Definition of API function CanIf_ReadTxNotifStatus ⌈

Service Name	CanIf_ReadTxNotifStatus	
Syntax	`CanIf_NotifStatusType CanIf_ReadTxNotifStatus ( PduIdType CanIfTxSduId )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CanIfTxSduId	L-SDU handle to be transmitted. This handle specifies the corresponding CAN L-SDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
Parameters (inout)	None	
Parameters (out)	None	
Return value	CanIf_NotifStatusType	Current confirmation status of the corresponding CAN Tx L-PDU.
Description	This service returns the confirmation status (confirmation occurred or not) of a specific static or dynamic CAN Tx L-PDU, requested by the CanIfTxSduId.	
Available via	CanIf.h	

⌋

Note: This function notifies the upper layer about any transmit confirmation event to the corresponding requested L-SDU.

[SWS_CANIF_00393] ⌈If configuration parameters CanIfPublicReadTxPduNotifyStatusApi and CanIfTxPduReadNotifyStatus for the transmitted L-SDU are set to TRUE, and if CanIf_ReadTxNotifStatus() is called, the CanIf shall reset the notification status for the transmitted L-SDU.⌋

[SWS_CANIF_00331]

Upstream requirements: SRS_BSW_00323

⌈If parameter CanIfTxSduId of CanIf_ReadTxNotifStatus() is out of range or if no status information was configured for this CAN Tx L-SDU, CanIf shall report development error code CANIF_E_INVALID_TXPDUID to the Det_ReportError service of the DET when CanIf_ReadTxNotifStatus() is called.⌋

[SWS_CANIF_00335] ⌈Configuration of CanIf_ReadTxNotifStatus(): This API can be enabled or disabled at pre-compile time configuration globally by the parameter CanIfPublicReadTxPduNotifyStatusApi.⌋

8.3.10 CanIf_ReadRxNotifStatus

[SWS_CANIF_00230] Definition of API function CanIf_ReadRxNotifStatus ⌈

Service Name	CanIf_ReadRxNotifStatus	
Syntax	`CanIf_NotifStatusType CanIf_ReadRxNotifStatus ( PduIdType CanIfRxSduId )`	
Service ID [hex]	0x08	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CanIfRxSduId	Receive L-SDU handle specifying the corresponding CAN L-SDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
Parameters (inout)	None	
Parameters (out)	None	
Return value	CanIf_NotifStatusType	Current indication status of the corresponding CAN Rx L-PDU.
Description	This service returns the indication status (indication occurred or not) of a specific CAN Rx L-PDU, requested by the CanIfRxSduId.	
Available via	CanIf.h	

⌋

Note: This function notifies the upper layer about any receive indication event to the corresponding requested L-SDU.

[SWS_CANIF_00394] ⌈If configuration parameters CanIfPublicReadRxPduNotifyStatusApi and CanIfRxPduReadNotifyStatus are set to TRUE, and if CanIf_ReadRxNotifStatus() is called, then CanIf shall reset the notification status for the received L-SDU.⌋

[SWS_CANIF_00336]

Upstream requirements: SRS_BSW_00323

⌈If parameter CanIfRxSduId of CanIf_ReadRxNotifStatus() is out of range or if status for CanRxPduId was requested whereas CanIfRxPduReadData is disabled or

if no status information was configured for this CAN Rx L-SDU, CanIf shall report development error code CANIF_E_INVALID_RXPDUID to the Det_ReportError service of the DET, when CanIf_ReadRxNotifStatus() is called.⌋

Note: The function CanIf_ReadRxNotifStatus() must not be used for CanIfRxSduIds, which are defined to receive multiple CAN-Ids (range reception).

[SWS_CANIF_00340] ⌈Configuration of CanIf_ReadRxNotifStatus(): This API can be enabled or disabled at pre-compile time configuration globally by the parameter CanIfPublicReadRxPduNotifyStatusApi.⌋

8.3.11 CanIf_SetPduMode

[SWS_CANIF_00008] Definition of API function CanIf_SetPduMode ⌈

Service Name	CanIf_SetPduMode	
Syntax	`Std_ReturnType CanIf_SetPduMode ( uint8 ControllerId, CanIf_PduModeType PduModeRequest )`	
Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ControllerId	All PDUs of the own ECU connected to the corresponding CanIf ControllerId, which is assigned to a physical CAN controller are addressed.
	PduModeRequest	Requested PDU mode change
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Request for mode transition has been accepted. E_NOT_OK: Request for mode transition has not been accepted.
Description	This service sets the requested mode at the L-PDUs of a predefined logical PDU channel.	
Available via	CanIf.h	

⌋

Note: The channel parameter denoting the predefined logical PDU channel can be derived from parameter ControllerId of function CanIf_SetPduMode().

[SWS_CANIF_00341]

Upstream requirements: SRS_BSW_00323

⌈If CanIf_SetPduMode() is called with invalid ControllerId, CanIf shall report development error code CANIF_E_PARAM_CONTROLLERID to the Det_ReportError service of the DET module.⌋

[SWS_CANIF_00860]

Upstream requirements: SRS_BSW_00323

⌈If `CanIf_SetPduMode()` is called with invalid `PduModeRequest`, `CanIf` shall report development error code `CANIF_E_PARAM_PDU_MODE` to the `Det_ReportError` service of the *DET* module.⌋

[SWS_CANIF_00874] ⌈The service `CanIf_SetPduMode()` shall not accept any request and shall return `E_NOT_OK`, if the controller mode referenced by `ControllerId` is not in state `CAN_CS_STARTED`.⌋

8.3.12 CanIf_GetPduMode

[SWS_CANIF_00009] Definition of API function CanIf_GetPduMode ⌈

Service Name	CanIf_GetPduMode	
Syntax	`Std_ReturnType CanIf_GetPduMode ( uint8 ControllerId, CanIf_PduModeType* PduModePtr )`	
Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Reentrant (Not for the same channel)	
Parameters (in)	ControllerId	All PDUs of the own ECU connected to the corresponding CanIf ControllerId, which is assigned to a physical CAN controller are addressed.
Parameters (inout)	None	
Parameters (out)	PduModePtr	Pointer to a memory location, where the current mode of the logical PDU channel will be stored.
Return value	Std_ReturnType	`E_OK`: PDU mode request has been accepted `E_NOT_OK`: PDU mode request has not been accepted
Description	This service reports the current mode of a requested PDU channel.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00346]

Upstream requirements: SRS_BSW_00323

⌈If `CanIf_GetPduMode()` is called with invalid `ControllerId`, `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module.⌋

[SWS_CANIF_00657]

Upstream requirements: SRS_BSW_00323

⌈If `CanIf_GetPduMode()` is called with invalid `PduModePtr`, `CanIf` shall report development error code `CANIF_E_PARAM_POINTER` to the `Det_ReportError` service of the DET module.⌋

8.3.13 CanIf_SetDynamicTxId

[SWS_CANIF_00189] Definition of API function CanIf_SetDynamicTxId ⌈

Service Name	CanIf_SetDynamicTxId	
Syntax	`void CanIf_SetDynamicTxId ( PduIdType CanIfTxSduId, Can_IdType CanId )`	
Service ID [hex]	0x0c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CanIfTxSduId	L-SDU handle to be transmitted. This handle specifies the corresponding CAN L-SDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
	CanId	Standard/Extended CAN ID of CAN L-SDU that shall be transmitted as FD or conventional CAN frame.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service reconfigures the corresponding CAN identifier of the requested CAN L-PDU.	
Available via	CanIf.h	

⌋

Note: `CanIf_SetDynamicTxId()` may be interrupted by `CanIf_Transmit()` called by several modules in the communication stack. Therefore precautions for preventing inconsistency need to be considered.

[SWS_CANIF_00944] ⌈If parameter `CanIfTxSduId` of `CanIf_SetDynamicTxId()` refers to a `CanIfTxPduCfg` with `CanIfTxPduXLParams`, `CanIf` shall report development error code `CANIF_E_INVALID_TXPDUID` to the `Det_ReportError` service of the DET module, when `CanIf_SetDynamicTxId()` is called.⌋

[SWS_CANIF_00352]

Upstream requirements: SRS_BSW_00323

⌈If parameter `CanIfTxSduId` of `CanIf_SetDynamicTxId()` has an invalid value, `CanIf` shall report development error code `CANIF_E_INVALID_TXPDUID` to the `Det_ReportError` service of the DET module, when `CanIf_SetDynamicTxId()` is called.⌋

[SWS_CANIF_00353]

Upstream requirements: SRS_BSW_00323

⌈If parameter `CanId` of `CanIf_SetDynamicTxId()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_CANID` to the `Det_ReportError` service of the DET module, when `CanIf_SetDynamicTxId()` is called.⌋

[SWS_CANIF_00355] ⌈If `CanIf` was not initialized before calling `CanIf_SetDynamicTxId()`, then the function `CanIf_SetDynamicTxId()` shall not execute a reconfiguration of Tx CanId.⌋

[SWS_CANIF_00357] ⌈Configuration of `CanIf_SetDynamicTxId()`: This function shall be pre compile time configurable `On/Off` by the configuration parameter `CanIf-PublicSetDynamicTxIdApi`.⌋

8.3.14 CanIf_SetTrcvMode

[SWS_CANIF_00287] Definition of API function CanIf_SetTrcvMode ⌈

Service Name	CanIf_SetTrcvMode	
Syntax	`Std_ReturnType CanIf_SetTrcvMode ( uint8 TransceiverId, CanTrcv_TrcvModeType TransceiverMode )`	
Service ID [hex]	0x0d	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for mode transition
	TransceiverMode	Requested mode transition
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Transceiver mode request has been accepted. `E_NOT_OK`: Transceiver mode request has not been accepted.
Description	This service changes the operation mode of the tansceiver TransceiverId, via calling the corresponding CAN Transceiver Driver service.	
Available via	CanIf.h	

⌋

Note: For more details, please refer to the [2, Specification of CAN Transceiver Driver].

[SWS_CANIF_00358] ⌈The function `CanIf_SetTrcvMode()` shall call the function `CanTrcv_SetOpMode(Transceiver, OpMode)` on the corresponding requested CAN Transceiver Driver module.⌋

Note: The parameters of the service `CanTrcv_SetOpMode()` are of type:

- `OpMode`: `CanTrcv_TrcvModeType`(desired operation mode)
- `Transceiver`: `uint8` (Transceiver to which function call has to be applied)

(see [2, Specification of CAN Transceiver Driver])

[SWS_CANIF_00538]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TransceiverId` of `CanIf_SetTrcvMode()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET, when `CanIf_SetTrcvMode()` is called.⌋

Note: The mode of a transceiver can only be changed to CANTRCV_TRCVMODE_STANDBY, when the former mode of the transceiver has been CANTRCV_TRCVMODE_NORMAL (see [2]). But this is not checked by the CanIf.

Note: The mode of a transceiver can only be changed to CANTRCV_TRCVMODE_SLEEP, when the former mode of the transceiver has been CANTRCV_TRCVMODE_STANDBY (see [2]). But this is not checked by the CanIf.

[SWS_CANIF_00648]

Upstream requirements: SRS_BSW_00323

⌈If parameter TransceiverMode of CanIf_SetTrcvMode() has an invalid value (not CANTRCV_TRCVMODE_STANDBY, CANTRCV_TRCVMODE_SLEEP or CANTRCV_TRCVMODE_NORMAL), the CanIf shall report development error code CANIF_E_PARAM_TRCVMODE to the Det_ReportError service of the DET module, when CanIf_SetTrcvMode() is called.⌋

Note: The function CanIf_SetTrcvMode() should be applicable to all CAN transceivers with all values of TransceiverMode independent, if the transceiver hardware supports these modes or not. This is to ease up the view of the CanIf to the assigned physical CAN channel.

[SWS_CANIF_00362] ⌈Configuration of CanIf_SetTrcvMode(): The number of supported transceiver types for each network is set up in the configuration phase (see CanIfTrcvCfg and CanIfTrcvDrvCfg). If no transceiver is used, this function may be omitted. Therefore, if no transceiver is configured in LT or PB class the API shall return with E_NOT_OK.⌋

8.3.15 CanIf_GetTrcvMode

[SWS_CANIF_00288] Definition of API function CanIf_GetTrcvMode ⌈

Service Name	CanIf_GetTrcvMode	
Syntax	`Std_ReturnType CanIf_GetTrcvMode ( uint8 TransceiverId, CanTrcv_TrcvModeType* TransceiverModePtr )`	
Service ID [hex]	0x0e	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for current operation mode.
Parameters (inout)	None	
Parameters (out)	TransceiverModePtr	Requested mode of requested network the Transceiver is connected to.
Return value	Std_ReturnType	E_OK: Transceiver mode request has been accepted. E_NOT_OK: Transceiver mode request has not been accepted.

▽

△

Description	This function invokes CanTrcv_GetOpMode and updates the parameter TransceiverModePtr with the value OpMode provided by CanTrcv.
Available via	CanIf.h

⌋

Note: For more details, please refer to the [2, Specification of CAN Transceiver Driver].

[SWS_CANIF_00363] ⌈The function `CanIf_GetTrcvMode()` shall call the function `CanTrcv_GetOpMode(Transceiver, OpMode)` on the corresponding requested CAN Transceiver Driver module.⌋

Note: The parameters of the function `CanTrcv_GetOpMode` are of type:

- `OpMode`: `CanTrcv_TrcvModeType` (desired operation mode)
- `Transceiver`: `uint8` (Transceiver to which API call has to be applied)

(see [2, Specification of CAN Transceiver Driver])

[SWS_CANIF_00364]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TransceiverId` of `CanIf_GetTrcvMode()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_GetTrcvMode()` is called.⌋

[SWS_CANIF_00650]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TransceiverModePtr` of `CanIf_GetTrcvMode()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_POINTER` to the `Det_ReportError` service of the DET module, when `CanIf_GetTrcvMode()` was called.⌋

[SWS_CANIF_00367] ⌈Configuration of `CanIf_GetTrcvMode()`: The number of supported transceiver types for each network is set up in the configuration phase (see `CanIfTrcvCfg` and `CanIfTrcvDrvCfg`). If no transceiver is used, this function may be omitted. Therefore, if no transceiver is configured in LT or PB class the API shall return with `E_NOT_OK`.⌋

8.3.16 CanIf_GetTrcvWakeupReason

[SWS_CANIF_00289] Definition of API function CanIf_GetTrcvWakeupReason ⌈

Service Name	CanIf_GetTrcvWakeupReason	
Syntax	`Std_ReturnType CanIf_GetTrcvWakeupReason ( uint8 TransceiverId, CanTrcv_TrcvWakeupReasonType* TrcvWuReasonPtr )`	
Service ID [hex]	0x0f	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for wake up reason.
Parameters (inout)	None	
Parameters (out)	TrcvWuReasonPtr	provided pointer to where the requested transceiver wake up reason shall be returned
Return value	Std_ReturnType	`E_OK`: Transceiver wake up reason request has been accepted. `E_NOT_OK`: Transceiver wake up reason request has not been accepted.
Description	This service returns the reason for the wake up of the transceiver TransceiverId, via calling the corresponding CAN Transceiver Driver service.	
Available via	CanIf.h	

⌋

Note: The ability to detect and differentiate the possible wake up reasons depends strongly on the CAN transceiver hardware. For more details, please refer to the [2, Specification of CAN Transceiver Driver].

[SWS_CANIF_00368] ⌈The function `CanIf_GetTrcvWakeupReason()` shall call `CanTrcv_GetBusWuReason(Transceiver, Reason)` on the corresponding requested `CanTrcv`.⌋

Note: The parameters of the function `CanTrcv_GetBusWuReason()` are of type:

- `reason`: `CanTrcv_TrcvWakeupReasonType`
- `Transceiver`: `uint8` (Transceiver to which API call has to be applied)

(see [2, Specification of CAN Transceiver Driver])

[SWS_CANIF_00537]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TransceiverId` of `CanIf_GetTrcvWakeupReason()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_GetTrcvWakeupReason()` is called.⌋

[SWS_CANIF_00649]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TrcvWuReasonPtr` of `CanIf_GetTrcvWakeupReason()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_POINTER` to the `Det_ReportError` service of the DET module, when `CanIf_GetTrcvWakeupReason()` is called.⌋

Note: Please be aware, that if more than one network is available, each network may report a different wake-up reason. E.g. if an ECU uses CAN, a wake-up by CAN may occur and the incoming data may cause an internal wake-up for another CAN network.

The service `CanIf_GetTrcvWakeupReason()` has a "per network" view and does not vote the more important reason or sequence internally. The same may be true if e.g. one transceiver controls the power supply and the other is just powered or un-powered. Then one may be able to return `CANTRCV_WU_POWER_ON`, whereas the other may state e.g. `CANTRCV_WU_RESET`. It is up to the calling module to decide, how to handle the wake-up information.

[SWS_CANIF_00371] ⌈Configuration of `CanIf_GetTrcvWakeupReason()`: The number of supported transceiver types for each network is set up in the configuration phase (see `CanIfTrcvCfg` and `CanIfTrcvDrvCfg`). If no transceiver is used, this function may be omitted. Therefore, if no transceiver is configured in LT or PB class the API shall return with `E_NOT_OK`.⌋

8.3.17 CanIf_SetTrcvWakeupMode

[SWS_CANIF_00290] Definition of API function CanIf_SetTrcvWakeupMode ⌈

Service Name	CanIf_SetTrcvWakeupMode	
Syntax	`Std_ReturnType CanIf_SetTrcvWakeupMode ( uint8 TransceiverId, CanTrcv_TrcvWakeupModeType TrcvWakeupMode )`	
Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for wake up notification mode transition.
	TrcvWakeupMode	Requested transceiver wake up notification mode
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	Std_ReturnType	E_OK: Will be returned, if the wake up notifications state has been changed to the requested mode. E_NOT_OK: Will be returned, if the wake up notifications state change has failed or the parameter is out of the allowed range. The previous state has not been changed.
Description	This function shall call CanTrcv_SetTrcvWakeupMode.	
Available via	CanIf.h	

⌋

Note: For more details, please refer to [2, Specification of CAN Transceiver Driver].

[SWS_CANIF_00372] ⌈The function `CanIf_SetTrcvWakeupMode()` shall call `CanTrcv_SetWakeupMode(Transceiver, TrcvWakeupMode)` on the corresponding requested `CanTrcv`.⌋

Info: The parameters of the function `CanTrcv_SetWakeupMode()` are of type:

- `TrcvWakeupMode`: `CanTrcv_TrcvWakeupModeType` (see [2, Specification of CAN Transceiver Driver])
- `Transceiver`: `uint8` (Transceiver to which API call has to be applied)

(see [2, Specification of CAN Transceiver Driver])

Note: The following three paragraphs are already described in the Specification of CanTrcv (see [2]). They describe the behavior of a `CanTrcv` in the respective transceiver wake-up mode, which is requested in parameter `TrcvWakeupMode`.

`CANTRCV_WUMODE_ENABLE`:
If the `CanTrcv` has a stored wake-up event pending for the addressed CanNetwork, the notification is executed within or immediately after the function `CanTrcv_SetWakeupMode()` (depending on the implementation).

`CANTRCV_WUMODE_DISABLE`:
No notifications for wake-up events for the addressed CanNetwork are passed through the `CanTrcv`. The transceiver device and the underlying communication driver has to buffer detected wake-up events and raise the event(s), when the wake-up notification is enabled again.

`CANTRCV_WUMODE_CLEAR`:
If notification of wake-up events is disabled (see description of mode `CANTRCV_WUMODE_DISABLE`), detected wake-up events are buffered. Calling `CanIf_SetTrcvWakeupMode()` with parameter `CANTRCV_WUMODE_CLEAR` clears these bufferd events. Clearing of wake-up events has to be used, when the wake-up notification is disabled to clear all stored wake-up events under control of the higher layers of the `CanTrcv`.

[SWS_CANIF_00535]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TransceiverId` of `CanIf_SetTrcvWakeupMode()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_SetTrcvWakeupMode()` is called.⌋

[SWS_CANIF_00536]

Upstream requirements: SRS_BSW_00323

⌈If parameter `TrcvWakeupMode` of `CanIf_SetTrcvWakeupMode()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCVWAKEUPMODE` to the `Det_ReportError` service of the DET module, when `CanIf_SetTrcvWakeupMode()` is called.⌋

[SWS_CANIF_00373] ⌈Configuration of `CanIf_SetTrcvWakeupMode()`: The number of supported transceiver types for each network is set up in the configuration phase (see `CanIfTrcvCfg` and `CanIfTrcvDrvCfg`). If no transceiver is used, this function may be omitted. Therefore, if no transceiver is configured in LT or PB class the API shall return with `E_NOT_OK`.⌋

8.3.18 CanIf_CheckWakeup

[SWS_CANIF_00219] Definition of API function CanIf_CheckWakeup ⌈

Service Name	CanIf_CheckWakeup	
Syntax	`Std_ReturnType CanIf_CheckWakeup ( EcuM_WakeupSourceType WakeupSource )`	
Service ID [hex]	0x11	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Parameters (in)	WakeupSource	Source device, which initiated the wake up event: CAN controller or CAN transceiver
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Will be returned, if the check wake up request has been accepted `E_NOT_OK`: Will be returned, if the check wake up request has not been accepted
Description	This service checks, whether an underlying CAN driver or a CAN transceiver driver already signals a wakeup event.	
Available via	CanIf.h	

⌋

Note: *Integration Code* calls this function

[SWS_CANIF_00398]

Upstream requirements: SRS_BSW_00323

⌈If parameter `WakeupSource` of `CanIf_CheckWakeup()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_WAKEUPSOURCE` to the `Det_ReportError` service of the DET, when `CanIf_CheckWakeup()` is called.⌋

Note: The call context of `CanIf_CheckWakeup()` is either on interrupt level (interrupt mode) or on task level (polling mode).

8.3.19 CanIf_CheckValidation

[SWS_CANIF_00178] Definition of API function CanIf_CheckValidation ⌈

Service Name	CanIf_CheckValidation	
Syntax	`Std_ReturnType CanIf_CheckValidation ( EcuM_WakeupSourceType WakeupSource )`	
Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	WakeupSource	Source device which initiated the wake-up event and which has to be validated: CAN controller or CAN transceiver
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Will be returned, if the check validation request has been accepted. `E_NOT_OK`: Will be returned, if the check validation request has not been accepted.
Description	This service is performed to validate a previous wakeup event.	
Available via	CanIf.h	

⌋

Note: *Integration Code* calls this function

[SWS_CANIF_00404]

Upstream requirements: SRS_BSW_00323

⌈If parameter `WakeupSource` of `CanIf_CheckValidation()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_WAKEUPSOURCE` to the `Det_ReportError` service of the DET module, when `CanIf_CheckValidation()` is called.⌋

Note: The call context of `CanIf_CheckValidation()` is either on interrupt level (interrupt mode) or on task level (polling mode).

Caveat: The corresponding CAN controller and transceiver must be switched on via `CanTrcv_SetOpMode(Transceiver, CANTRCV_TRCVMODE_NORMAL)` and `Can_SetControllerMode(Controller, CAN_CS_STARTED)` and the corresponding mode indications must have been called.

[SWS_CANIF_00408] ⌈Configuration of `CanIf_CheckValidation()`: If no validation is needed, this API can be omitted by disabling of `CanIfPublicWakeupCheckValidSupport`.⌋

8.3.20 CanIf_GetTxConfirmationState

[SWS_CANIF_00734] Definition of API function CanIf_GetTxConfirmationState ⌈

Service Name	CanIf_GetTxConfirmationState	
Syntax	`CanIf_NotifStatusType CanIf_GetTxConfirmationState ( uint8 ControllerId )`	
Service ID [hex]	0x19	
Sync/Async	Synchronous	
Reentrancy	Reentrant (Not for the same controller)	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout)	None	
Parameters (out)	None	
Return value	CanIf_NotifStatusType	Combined TX confirmation status for all TX PDUs of the CAN controller
Description	This service reports, if any TX confirmation has been done for the whole CAN controller since the last CAN controller start.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00736] ⌈If parameter `ControllerId` of `CanIf_GetTxConfirmationState()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module, when `CanIf_GetTxConfirmationState()` is called.⌋

Note: The call context of `CanIf_GetTxConfirmationState()` is on task level (polling mode).

[SWS_CANIF_00738] ⌈Configuration of `CanIf_GetTxConfirmationState()`: If BusOff Recovery of CanSm doesn't need the status of the Tx confirmations (see [SWS_CANIF_00740]), this API can be omitted by disabling of `CanIfPublicTxConfirmPollingSupport`.⌋

8.3.21 CanIf_ClearTrcvWufFlag

[SWS_CANIF_00760] Definition of API function CanIf_ClearTrcvWufFlag ⌈

Service Name	CanIf_ClearTrcvWufFlag	
Syntax	`Std_ReturnType CanIf_ClearTrcvWufFlag ( uint8 TransceiverId )`	
Service ID [hex]	0x1e	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different CAN transceivers	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to the designated CAN transceiver.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request has been accepted `E_NOT_OK`: Request has not been accepted
Description	Requests the CanIf module to clear the WUF flag of the designated CAN transceiver.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00766] ⌈Within `CanIf_ClearTrcvWufFlag()` the function `CanTrcv_ClearTrcvWufFlag()` shall be called.⌋

[SWS_CANIF_00769] ⌈If parameter `TransceiverId` of `CanIf_ClearTrcvWufFlag()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_ClearTrcvWufFlag()` is caled.⌋

[SWS_CANIF_00771] ⌈Configuration of `CanIf_ClearTrcvWufFlag()`: Whether the CanIf supports this function shall be pre compile time configurable `On/Off` by the configuration parameter `CanIfPublicPnSupport` and `CanIfPublicTrcvPnEnable`.⌋

8.3.22 CanIf_CheckTrcvWakeFlag

[SWS_CANIF_00761] Definition of API function CanIf_CheckTrcvWakeFlag ⌈

Service Name	CanIf_CheckTrcvWakeFlag	
Syntax	`Std_ReturnType CanIf_CheckTrcvWakeFlag ( uint8 TransceiverId )`	
Service ID [hex]	0x1f	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different CAN transceivers	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to the designated CAN transceiver.

▽

△

Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Request has been accepted `E_NOT_OK`: Request has not been accepted
Description	Requests the CanIf module to check the Wake flag of the designated CAN transceiver.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00765] ⌈Within `CanIf_CheckTrcvWakeFlag()` the function `CanTrcv_CheckWakeFlag()` shall be called.⌋

[SWS_CANIF_00770] ⌈If parameter `TransceiverId` of `CanIf_CheckTrcvWakeFlag()` has an invalid value, the CanIf shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_CheckTrcvWakeFlag()` is caled.⌋

[SWS_CANIF_00813] ⌈Configuration of `CanIf_CheckTrcvWakeFlag()`: Whether the CanIf supports this function shall be pre compile time configurable `On/Off` by the configuration parameter `CanIfPublicPnSupport` and `CanIfPublicTrcvPnEnable`.⌋

8.3.23 CanIf_SetBaudrate

[SWS_CANIF_00867] Definition of API function CanIf_SetBaudrate ⌈

Service Name	CanIf_SetBaudrate	
Syntax	`Std_ReturnType CanIf_SetBaudrate ( uint8 ControllerId, uint16 BaudRateConfigID )`	
Service ID [hex]	0x27	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerIds. Non reentrant for the same ControllerId.	
Parameters (in)	ControllerId	Abstract CanIf ControllerId which is assigned to a CAN controller, whose baud rate shall be set.
	BaudRateConfigID	references a baud rate configuration by ID (see CanController BaudRateConfigID)
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Service request accepted, setting of (new) baud rate started `E_NOT_OK`: Service request not accepted
Description	This service shall set the baud rate configuration of the CAN controller. Depending on necessary baud rate modifications the controller might have to reset.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00868] ⌈The service CanIf_SetBaudrate() shall call Can_SetBaudrate(Controller, BaudRateConfigID) for the requested CAN Controller.⌋

[SWS_CANIF_00869]

Upstream requirements: SRS_BSW_00323

⌈If CanIf_SetBaudrate() is called with invalid ControllerId, CanIf shall report development error code CANIF_E_PARAM_CONTROLLERID to the Det_ReportError service of the DET module.⌋

Note: The parameter BaudRateConfigID of CanIf_SetBaudrate() is not checked by CanIf. This has to be done by responsible CanDrv.

Note: The call context of CanIf_SetBaudrate() is on task level (polling mode).

[SWS_CANIF_00871] ⌈If CanIf supports changing baud rate and thus CanIf_SetBaudrate(), shall be configurable via CanIfSetBaudrateApi.⌋

8.3.24 CanIf_GetControllerRxErrorCounter

[SWS_CANIF_91003] Definition of API function CanIf_GetControllerRxErrorCounter ⌈

Service Name	CanIf_GetControllerRxErrorCounter	
Syntax	`Std_ReturnType CanIf_GetControllerRxErrorCounter ( uint8 ControllerId, uint8* RxErrorCounterPtr )`	
Service ID [hex]	0x4d	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller.
Parameters (inout)	None	
Parameters (out)	RxErrorCounterPtr	Pointer to a memory location, where the current Rx error counter of the CAN controller will be stored.
Return value	Std_ReturnType	E_OK: Rx error counter available. E_NOT_OK: Wrong ControllerId, or Rx error counter not available.
Description	This service calls the corresponding CAN Driver service for obtaining the Rx error counter of the CAN controller.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00907]

Upstream requirements: SRS_BSW_00323

⌈If parameter ControllerId of CanIf_GetControllerRxErrorCounter() has an invalid value, the CanIf shall report development error code CANIF_E_PARAM_CONTROLLERID to the Det_ReportError service of the DET, when CanIf_GetControllerRxErrorCounter() is called.⌋

[SWS_CANIF_00908]

Upstream requirements: SRS_BSW_00323

⌈If parameter RxErrorCounterPtr of CanIf_GetControllerRxErrorCounter() is a null pointer, the CanIf shall report development error code CANIF_E_PARAM_POINTER to the Det_ReportError service of the DET, when CanIf_GetControllerRxErrorCounter() is called.⌋

8.3.25 CanIf_GetControllerTxErrorCounter

[SWS_CANIF_91004] Definition of API function CanIf_GetControllerTxErrorCounter ⌈

Service Name	CanIf_GetControllerTxErrorCounter	
Syntax	`Std_ReturnType CanIf_GetControllerTxErrorCounter ( uint8 ControllerId, uint8* TxErrorCounterPtr )`	
Service ID [hex]	0x4e	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller.
Parameters (inout)	None	
Parameters (out)	TxErrorCounterPtr	Pointer to a memory location, where the current Tx error counter of the CAN controller will be stored.
Return value	Std_ReturnType	E_OK: Tx error counter available. E_NOT_OK: Wrong ControllerId, or Tx error counter not available.
Description	This service calls the corresponding CAN Driver service for obtaining the Tx error counter of the CAN controller.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00909]

Upstream requirements: SRS_BSW_00323

⌈If parameter ControllerId of CanIf_GetControllerTxErrorCounter() has an invalid value, the CanIf shall report development error code CANIF_E_PARAM_CONTROLLERID to the Det_ReportError service of the DET, when CanIf_GetControllerTxErrorCounter() is called.⌋

[SWS_CANIF_00910]

Upstream requirements: SRS_BSW_00323

⌈If parameter TxErrorCounterPtr of CanIf_GetControllerTxErrorCounter() is a null pointer, the CanIf shall report development error code CANIF_E_PARAM_POINTER to the Det_ReportError service of the DET, when CanIf_GetControllerTxErrorCounter() is called.⌋

8.3.26 CanIf_EnableBusMirroring

[SWS_CANIF_91005] Definition of API function CanIf_EnableBusMirroring ⌈

Service Name	CanIf_EnableBusMirroring	
Syntax	`Std_ReturnType CanIf_EnableBusMirroring ( uint8 ControllerId, boolean MirroringActive )`	
Service ID [hex]	0x4c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller.
	MirroringActive	TRUE: Mirror_ReportCanFrame will be called for each frame received or transmitted on the given controller. FALSE: Mirror_ReportCanFrame will not be called for the given controller.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Mirroring mode was changed. `E_NOT_OK`: Wrong ControllerId, or mirroring globally disabled (see CanIfBusMirroringSupport).
Description	Enables or disables mirroring for a CAN controller.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00911] ⌈If Bus Mirroring is not enabled (see `CanIfBusMirroringSupport`), the API `CanIf_EnableBusMirroring()` can be omitted.⌋

[SWS_CANIF_00912]

Upstream requirements: SRS_BSW_00323

⌈If parameter `ControllerId` of `CanIf_EnableBusMirroring()` has an invalid value, the `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET, when `CanIf_EnableBusMirroring()` is called.⌋

8.3.27 CanIf_GetCurrentTime

[SWS_CANIF_91014] Definition of API function CanIf_GetCurrentTime

Status: DRAFT

⌈

Service Name	CanIf_GetCurrentTime (draft)
Syntax	`Std_ReturnType CanIf_GetCurrentTime ( uint8 Controller, Can_TimeStampType* timeStampPtr )`

▽

△

Service ID [hex]	0x51	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Controller	Index of the addresses CAN controller.
Parameters (inout)	None	
Parameters (out)	timeStampPtr	current time stamp
Return value	Std_ReturnType	`E_OK`: successful `E_NOT_OK`: failed
Description	This service calls the corresponding CAN Driver service to retrieve the current time value out of the HW registers. **Tags:** atp.Status=draft	
Available via	CanIf.h	

⌋

[SWS_CANIF_00922]

Status: DRAFT

⌈If development error detection is enabled: the function shall check that the service `CanIf_Init()` was previously called. If the check fails, the function shall raise the development error `CANIF_E_UNINIT`⌋

[SWS_CANIF_00923]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `Controller` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_CONTROLLERID`.⌋

[SWS_CANIF_00924]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `timeStampPtr` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_POINTER`.⌋

[SWS_CANIF_00925]

Status: DRAFT

⌈The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`⌋

8.3.28 CanIf_EnableEgressTimeStamp

[SWS_CANIF_91011] Definition of API function CanIf_EnableEgressTimeStamp

Status: DRAFT

⌈

Service Name	CanIf_EnableEgressTimeStamp (draft)	
Syntax	`void CanIf_EnableEgressTimeStamp ( PduIdType TxPduId )`	
Service ID [hex]	0x52	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TxPduId	L-PDU handle of CAN L-PDU for which the time stamping shall be enabled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service calls the corresponding CAN Driver service to activate egress time stamping on a dedicated message object. **Tags:** atp.Status=draft	
Available via	CanIf.h	

⌋

[SWS_CANIF_00926]

Status: DRAFT

⌈If development error detection is enabled: the function shall check that the service `CanIf_Init()` was previously called. If the check fails, the function shall raise the development error `CANIF_E_UNINIT`⌋

[SWS_CANIF_00927]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `TxPduId` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_LPDU`.⌋

[SWS_CANIF_00928]

Status: DRAFT

⌈The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`⌋

8.3.29 CanIf_GetEgressTimeStamp

[SWS_CANIF_91012] Definition of API function CanIf_GetEgressTimeStamp

Status: DRAFT

⌈

Service Name	CanIf_GetEgressTimeStamp (draft)	
Syntax	`Std_ReturnType CanIf_GetEgressTimeStamp ( PduIdType TxPduId, Can_TimeStampType* timeStampPtr )`	
Service ID [hex]	0x53	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same TxPduId, Reentrant for different TxPduId	
Parameters (in)	TxPduId	L-PDU handle of CAN L-PDU for which the time stamp shall be returned.
Parameters (inout)	None	
Parameters (out)	timeStampPtr	current time stamp
Return value	Std_ReturnType	`E_OK`: successful `E_NOT_OK`: failed
Description	This service calls the corresponding CAN Driver service to read back the egress time stamp on a dedicated message object. It needs to be called within the TxConfirmation() function. **Tags:** atp.Status=draft	
Available via	CanIf.h	

⌋

[SWS_CANIF_00929]

Status: DRAFT

⌈If development error detection is enabled: the function shall check that the service `CanIf_Init()` was previously called. If the check fails, the function shall raise the development error `CANIF_E_UNINIT`⌋

[SWS_CANIF_00930]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `TxPduId` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_LPDU`.⌋

[SWS_CANIF_00931]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `timeStampPtr` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_POINTER`.⌋

[SWS_CANIF_00932]

Status: DRAFT

⌈The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`⌋

8.3.30 CanIf_GetIngressTimeStamp

[SWS_CANIF_91013] Definition of API function CanIf_GetIngressTimeStamp

Status: DRAFT

⌈

Service Name	CanIf_GetIngressTimeStamp (draft)	
Syntax	`Std_ReturnType CanIf_GetIngressTimeStamp ( PduIdType RxPduId, Can_TimeStampType timeStampPtr )`	
Service ID [hex]	0x54	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same RxPduId, Reentrant for different RxPduIds	
Parameters (in)	RxPduId	ID of the received I-PDU for which the time stamp shall be returned.
Parameters (inout)	None	
Parameters (out)	timeStampPtr	current time stamp
Return value	Std_ReturnType	`E_OK`: successful `E_NOT_OK`: failed
Description	This service calls the corresponding CAN Driver service to reads back the ingress time stamp on a dedicated message object. It needs to be called within the RxIndication() function. **Tags:** atp.Status=draft	
Available via	CanIf.h	

⌋

[SWS_CANIF_00933]

Status: DRAFT

⌈If development error detection is enabled: the function shall check that the service `CanIf_Init()` was previously called. If the check fails, the function shall raise the development error `CANIF_E_UNINIT`⌋

[SWS_CANIF_00934]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `RxPduId` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_LPDU`.⌋

[SWS_CANIF_00935]

Status: DRAFT

⌈If development error detection is enabled: the function shall check the parameter `timeStampPtr` for being valid. If the check fails, the function shall raise the development error `CANIF_E_PARAM_POINTER`.⌋

[SWS_CANIF_00936]

Status: DRAFT

⌈The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`⌋

8.3.31 CanIf_TriggerTransmit

[SWS_CANIF_00883] Definition of callback function CanIf_TriggerTransmit ⌈

Service Name	CanIf_TriggerTransmit	
Syntax	`Std_ReturnType CanIf_TriggerTransmit ( PduIdType TxPduId, PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x41	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	ID of the SDU that is requested to be transmitted.
Parameters (inout)	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLengh. On return, the service will indicate the length of the copied SDU data in SduLength.
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: SDU has been copied and SduLength indicates the number of copied bytes. `E_NOT_OK`: No SDU data has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Description	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00884] ⌈`CanIf` shall only provide the API function `CanIf_TriggerTransmit()` if TriggerTransmit support is enabled (`CanIfTriggerTransmitSupport = TRUE`).⌋

[SWS_CANIF_00885] ⌈The function `CanIf_TriggerTransmit()` shall call the `LSduR_CanIfTriggerTransmit()` function, passing the translated `TxPduId` and the pointer to the `PduInfo` structure (`PduInfoPtr`). Upon return, `CanIf_TriggerTransmit()` shall return the return value of its `LSduR_CanIfTriggerTransmit()`.⌋

8.3.32 CanIf_TxConfirmation

[SWS_CANIF_00007] Definition of callback function CanIf_TxConfirmation ⌈

Service Name	CanIf_TxConfirmation	
Syntax	`void CanIf_TxConfirmation ( PduIdType CanTxPduId )`	
Service ID [hex]	0x13	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	CanTxPduId	L-PDU handle of CAN L-PDU successfully transmitted. This ID specifies the corresponding CAN L-PDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service confirms a previously successfully processed transmission of a CAN TxPDU.	
Available via	CanIf.h	

⌋

Note: The service `CanIf_TxConfirmation()` is implemented in `CanIf` and called by the `CanDrv` after the `CAN L-PDU` has been transmitted on the CAN network.

Note: Due to the fact `CanDrv` does not support the HandleId concept as described in [19, Specification of ECU Configuration]: Within the service `CanIf_TxConfirmation()`, `CanDrv` uses `PduInfo->swPduHandle` as `CanTxPduId`, which was preserved from `Can_Write(Hth, *PduInfo)` or `CanXL_Write(Hth, *PduInfo)`.

[SWS_CANIF_00391] ⌈If configuration parameters `CanIfPublicReadTxPduNotifyStatusApi` and `CanIfTxPduReadNotifyStatus` for the `Transmitted L-PDU` are set to `TRUE`, and if `CanIf_TxConfirmation()` is called, `CanIf` shall set the notification status for the `Transmitted L-PDU`.⌋

[SWS_CANIF_00410]

Upstream requirements: SRS_BSW_00323

⌈If parameter `CanTxPduId` of `CanIf_TxConfirmation()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_LPDU` to the `Det_ReportError` service of the DET module, when `CanIf_TxConfirmation()` is called.⌋

[SWS_CANIF_00412] ⌈If `CanIf` was not initialized before calling `CanIf_TxConfirmation()`, `CanIf` shall not call the service `LSduR_CanIfTxConfirmation()` and shall not set the Tx confirmation status, when `CanIf_TxConfirmation()` is called.⌋

Note: The call context of `CanIf_TxConfirmation()` is either on interrupt level (interrupt mode) or on task level (polling mode).

8.3.33 CanIf_RxIndication

[SWS_CANIF_00006] Definition of callback function CanIf_RxIndication ⌈

Service Name	CanIf_RxIndication	
Syntax	`void CanIf_RxIndication ( const Can_HwType* Mailbox, const PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x14	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Mailbox	Identifies the HRH and its corresponding CAN Controller
	PduInfoPtr	Pointer to the received L-PDU
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates a successful reception of a received CAN Rx L-PDU to the CanIf after passing all filters and validation checks.	
Available via	CanIf.h	

⌋

Note: The service `CanIf_RxIndication()` is implemented in `CanIf` and called by `CanDrv` after a `CAN L-PDU` has been received.

[SWS_CANIF_00415] ⌈Within the service `CanIf_RxIndication()` the `CanIf` routes this indication to the configured upper layer target service(s).⌋

[SWS_CANIF_00392] ⌈If configuration parameters `CanIfPublicReadRxPduNotifyStatusApi` and `CanIfRxPduReadNotifyStatus` for the `Received L-PDU` are set to `TRUE`, and if `CanIf_RxIndication()` is called, the `CanIf` shall set the notification status for the `Received L-PDU`.⌋

[SWS_CANIF_00416]

Upstream requirements: SRS_BSW_00323

⌈If parameter `Mailbox->Hoh` of `CanIf_RxIndication()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_HOH` to the `Det_ReportError` service of the DET module, when `CanIf_RxIndication()` is called.⌋

[SWS_CANIF_00417]

Upstream requirements: SRS_BSW_00323

⌈If parameter `Mailbox->CanId` of `CanIf_RxIndication()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_CANID` to the `Det_ReportError` service of the DET module, when `CanIf_RxIndication()` is called.⌋

Note: If CanIf_RxIndication() is called with invalid PduInfoPtr->SduLength, runtime error CANIF_E_INVALID_DATA_LENGTH is reported (see [SWS_CANIF_00168]).

[SWS_CANIF_00419]

Upstream requirements: SRS_BSW_00323

⌈If parameter PduInfoPtr or Mailbox of CanIf_RxIndication() has an invalid value, CanIf shall report development error code CANIF_E_PARAM_POINTER to the Det_ReportError service of the DET module, when CanIf_RxIndication() is called.⌋

[SWS_CANIF_00421] ⌈If CanIf was not initialized before calling CanIf_RxIndication(), CanIf shall not execute *Rx indication handling*, when CanIf_RxIndication(), is called.⌋

Note: The call context of CanIf_RxIndication() is either on interrupt level (interrupt mode) or on task level (polling mode).

8.3.34 CanIf_XLRxIndication

[SWS_CANIF_91015] Definition of API function CanIf_XLRxIndication ⌈

Service Name	CanIf_XLRxIndication	

Syntax	`void CanIf_XLRxIndication (`
	`  const CanXL_HwType* Mailbox,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID [hex]	0x55	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Mailbox	Identifies the HRH and its corresponding CAN Controller
	PduInfoPtr	Pointer to the received L-PDU
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates a successful reception of a received CAN XL Rx L-PDU to the CanIf after passing all filters and validation checks. It provides the CAN XL specific parameters besides the hardware and actual L-PDU information.	
Available via	CanIf.h	

⌋

Note: The service CanIf_XLRxIndication() is implemented in CanIf and called by CAN XL Driver after a CAN XL L-PDU has been received. Depending on received SDU Type this might indicate a content based CAN XL frame (SDU Type 01h) or a tunneled CAN 2.0/FD frame (SDU Type 03h) or any arbitratry SDU Type handled by a CDD. An exception to this are mapped tunneled 802.3 Ethernet frames (SDU Type 05h) which are already handled by CAN XL Driver itself and therefore not passed

through to CanIf. Please note that unmapped tunneled 802.3 Ethernet frames (SDU Type 04h) are not handled by CAN XL Driver.

Note: The call context of CanIf_XLRxIndication() is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00945] ⌈Within the service CanIf_XLRxIndication() the CanIf routes this indication to the configured upper layer target service(s).⌋

[SWS_CANIF_00946] ⌈In case received SDU Type equals a content based CAN XL frame (SDU Type 01h) the matching CanIfRxPduCfg shall be searched. In case it is found, it shall be processed as configured, otherwise it shall be dropped.⌋

[SWS_CANIF_00947] ⌈In case received SDU Type equals a tunneled CAN 2.0/FD frame (Sdu Type 03h), CAN ID, DLC and payload shall be extracted and then handled as defined in CanIf_RxIndication() after matching the the Priority ID and the VCID as described in [SWS_CANIF_00952].⌋

[SWS_CANIF_00948] ⌈In case received SDU Type does neither equals a content based CAN XL frame (SDU Type 01h) or a tunneled CAN 2.0/FD frame (Sdu Type 03h), it shall not be further processed within CanIf.⌋

[SWS_CANIF_00949] ⌈If parameter Mailbox->Hoh of CanIf_XLRxIndication() has an invalid value, CanIf shall report development error code CANIF_E_PARAM_HOH to the Det_ReportError service of the DET module, when CanIf_XLRxIndication() is called.⌋

[SWS_CANIF_00950] ⌈If parameter PduInfoPtr, Mailbox or Mailbox->XLParams of CanIf_XLRxIndication() has an invalid value, CanIf shall report development error code CANIF_E_PARAM_POINTER to the Det_ReportError service of the DET module, when CanIf_RxIndication() is called.⌋

8.3.35 CanIf_XLReportErrorState

[SWS_CANIF_91018] Definition of API function CanIf_XLReportErrorState

Upstream requirements: SRS_BSW_00310, SRS_BSW_00369, SRS_BSW_00357, SRS_BSW_00406

⌈

Service Name	CanIf_XLReportErrorState
Syntax	`Std_ReturnType CanIf_XLReportErrorState ( uint8 Transceiver, Can_ErrorStateType ErrorState )`
Service ID [hex]	0x30
Sync/Async	Synchronous

▽

△

Reentrancy	Non Reentrant	
Parameters (in)	Transceiver	CAN transceiver to which API call has to be applied
	ErrorState	New error state of the corresponding CAN controller
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Report was successful `E_NOT_OK`: Call was rejected
Description	Reports each change of the CAN error state.	
Available via	CanIf.h	

⌋

Note: The service `CanIf_XLReportErrorState()` is implemented in `CanIf` and called by `CAN XL Driver` to report changes of the error state.

Note: The call context of `CanIf_XLReportErrorState()` is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00968] ⌈When `CanIf_XLReportErrorState()` is called, the `CanIf` shall in turn call `CanXLTrcv_ReportErrorState()`, and return the result returned by that call.⌋

[SWS_CANIF_00969] ⌈If development error detection is enabled: `CanIf_XLReportErrorState()` shall check that the service `CanIf_Init()` was previously called. If the check fails, the function shall raise the development error `CANIF_E_UNINIT`.⌋

Note: The parameters `Transceiver` and `ErrorState` will be checked by the `CAN XL Transceiver Driver`, an additional check in `CanIf` is not required.

8.3.36 CanIf_ControllerBusOff

[SWS_CANIF_00218] Definition of callback function CanIf_ControllerBusOff ⌈

Service Name	CanIf_ControllerBusOff	
Syntax	`void CanIf_ControllerBusOff ( uint8 ControllerId )`	
Service ID [hex]	0x16	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	Abstract CanIf ControllerId which is assigned to a CAN controller, where a BusOff occured.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates a Controller BusOff event referring to the corresponding CAN Controller with the abstract CanIf ControllerId.	

▽

△

Available via	CanIf.h

⌋

Note: The callback service `CanIf_ControllerBusOff()` is called by `CanDrv` and implemented in `CanIf`. It is called in case of a mode change notification of the `CanDrv`.

[SWS_CANIF_00429]

Upstream requirements: SRS_BSW_00323

⌈If parameter `ControllerId` of `CanIf_ControllerBusOff()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module, when `CanIf_ControllerBusOff()` is called.⌋

[SWS_CANIF_00431] ⌈If `CanIf` was not initialized before calling `CanIf_ControllerBusOff()`, `CanIf` shall not execute *BusOff notification*, when `CanIf_ControllerBusOff()`, is called.⌋

Note: The call context of `CanIf_ControllerBusOff()` is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00433] ⌈Configuration of `CanIf_ControllerBusOff()`: `ID` of the `CAN Controller` is published inside the configuration description of the `CanIf` (see `CanIfCtrlCfg`).⌋

Note: This service always has to be available, so there does not exist an appropriate configuration parameter.

8.3.37 CanIf_ConfirmPnAvailability

[SWS_CANIF_00815] Definition of callback function CanIf_ConfirmPnAvailability ⌈

Service Name	CanIf_ConfirmPnAvailability	
Syntax	`void CanIf_ConfirmPnAvailability ( uint8 TransceiverId )`	
Service ID [hex]	0x1a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, which was checked for PN availability.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	None
Description	This service indicates that the transceiver is running in PN communication mode referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.
Available via	CanIf.h

⌋

[SWS_CANIF_00753] ⌈If `CanIf_ConfirmPnAvailability()` is called, `CanIf` calls `<User_ConfirmPnAvailability>()`.⌋

Note: `CanIf` passes the delivered parameter `TransceiverId` to the upper layer module.

[SWS_CANIF_00816] ⌈If parameter `TransceiverId` of `CanIf_ConfirmPnAvailability()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_ConfirmPnAvailability()` is called.⌋

[SWS_CANIF_00817] ⌈If `CanIf` was not initialized before calling `CanIf_ConfirmPnAvailability()`, `CanIf` shall not execute notification, when `CanIf_ConfirmPnAvailability()` is called.⌋

Note: The call context of `CanIf_ConfirmPnAvailability()` is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00754] ⌈Configuration of `CanIf_ConfirmPnAvailability()`: This function shall be pre compile time configurable `ON`/`OFF` by the configuration parameter `CanIfPublicPnSupport` and `CanIfPublicTrcvPnEnable`.⌋

8.3.38 CanIf_ConfirmCtrlPnAvailability

[SWS_CANIF_91016] Definition of callback function CanIf_ConfirmCtrlPnAvailability

Status: DRAFT

⌈

Service Name	CanIf_ConfirmCtrlPnAvailability (draft)	
Syntax	`void CanIf_ConfirmCtrlPnAvailability ( uint8 ControllerId )`	
Service ID [hex]	0x56	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	Abstract CanIf ControllerId, which is assigned to a CAN Controller, which was checked for PN availability.

▽

△

Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This service indicates that the controller is running in PN communication mode referring to the corresponding CAN controller with the abstract CanIf ControllerId. **Tags:** atp.Status=draft
Available via	CanIf.h

⌋

[SWS_CANIF_00963] ⌈If `CanIf_ConfirmCtrlPnAvailability()` is called, `CanIf` calls `<User_ConfirmCtrlPnAvailability>()`.⌋

Note: `CanIf` passes the delivered parameter `ControllerId` to the upper layer module.

[SWS_CANIF_00964] ⌈If parameter `ControllerId` of `CanIf_ConfirmCtrlPnAvailability()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module, when `CanIf_ConfirmCtrlPnAvailability()` is called.⌋

[SWS_CANIF_00965] ⌈If `CanIf` was not initialized before calling `CanIf_ConfirmCtrlPnAvailability()`, `CanIf` shall not execute notification, when `CanIf_ConfirmCtrlPnAvailability()` is called.⌋

[SWS_CANIF_00966] ⌈Configuration of `CanIf_ConfirmCtrlPnAvailability()`: This function shall be pre compile time configurable `ON/OFF` by the configuration parameter `CanIfPublicPnSupport` and `CanIfPublicCtrlPnEnable`.⌋

8.3.39 CanIf_ClearTrcvWufFlagIndication

[SWS_CANIF_00762] Definition of callback function CanIf_ClearTrcvWufFlagIndication ⌈

Service Name	CanIf_ClearTrcvWufFlagIndication	
Syntax	`void CanIf_ClearTrcvWufFlagIndication ( uint8 TransceiverId )`	
Service ID [hex]	0x20	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, for which this function was called.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	This service indicates that the transceiver has cleared the WufFlag referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.
Available via	CanIf.h

⌋

[SWS_CANIF_00757] ⌈If `CanIf_ClearTrcvWufFlagIndication()` is called, `CanIf` calls `<User_ClearTrcvWufFlagIndication>()`.⌋

Note: `CanIf` passes the delivered parameter `TransceiverId` to the upper layer module.

[SWS_CANIF_00805] ⌈If parameter `TransceiverId` of `CanIf_ClearTrcvWufFlagIndication()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_ClearTrcvWufFlagIndication()` is called.⌋

[SWS_CANIF_00806] ⌈If `CanIf` was not initialized before calling `CanIf_ClearTrcvWufFlagIndication()`, `CanIf` shall not execute notification, when `CanIf_ClearTrcvWufFlagIndication()` is called.⌋

Note: The call context of `CanIf_ClearTrcvWufFlagIndication()` is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00808] ⌈Configuration of `CanIf_ClearTrcvWufFlagIndication()`: This function shall be pre compile time configurable `ON/OFF` by the configuration parameter `CanIfPublicPnSupport` and `CanIfPublicTrcvPnEnable`.⌋

8.3.40 CanIf_CheckTrcvWakeFlagIndication

[SWS_CANIF_00763] Definition of callback function CanIf_CheckTrcvWakeFlagIndication ⌈

Service Name	CanIf_CheckTrcvWakeFlagIndication	
Syntax	`void CanIf_CheckTrcvWakeFlagIndication ( uint8 TransceiverId )`	
Service ID [hex]	0x21	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, for which this function was called.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	This service indicates that the check of the transceiver's wake-up flag has been finished by the corresponding CAN transceiver with the abstract CanIf TransceiverId. This indication is used to cope with the asynchronous transceiver communication.
Available via	CanIf.h

⌋

[SWS_CANIF_00759] ⌈If `CanIf_CheckTrcvWakeFlagIndication()` is called, `CanIf` calls `<User_CheckTrcvWakeFlagIndication>()`.⌋

Note: `CanIf` passes the delivered parameter `TransceiverId` to the upper layer module.

[SWS_CANIF_00809] ⌈If parameter `TransceiverId` of `CanIf_CheckTrcvWakeFlagIndication()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_TRCV` to the `Det_ReportError` service of the DET module, when `CanIf_CheckTrcvWakeFlagIndication()` is called.⌋

[SWS_CANIF_00810] ⌈If the CanIf was not initialized before calling `CanIf_CheckTrcvWakeFlagIndication()`, `CanIf` shall not execute notification, when `CanIf_CheckTrcvWakeFlagIndication()` is called.⌋

Note: The call context of `CanIf_CheckTrcvWakeFlagIndication()` is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00812] ⌈Configuration of `CanIf_CheckTrcvWakeFlagIndication()`: This function shall be pre compile time configurable `ON/OFF` by the configuration parameter `CanIfPublicPnSupport` and `CanIfPublicTrcvPnEnable`.⌋

8.3.41 CanIf_ControllerModeIndication

[SWS_CANIF_00699] Definition of callback function CanIf_ControllerModeIndication ⌈

Service Name	CanIf_ControllerModeIndication	
Syntax	`void CanIf_ControllerModeIndication ( uint8 ControllerId, Can_ControllerStateType ControllerMode )`	
Service ID [hex]	0x17	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	Abstract CanIf ControllerId which is assigned to a CAN controller, which state has been transitioned.
	ControllerMode	Mode to which the CAN controller transitioned
Parameters (inout)	None	

▽

△

Parameters (out)	None
Return value	None
Description	This service indicates a controller state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId.
Available via	CanIf.h

⌋

Note: The callback service `CanIf_ControllerModeIndication()` is called by `CanDrv` and implemented in `CanIf`. It is called in case of a state transition notification of the `CanDrv`.

[SWS_CANIF_00700] ⌈If parameter `ControllerId` of `CanIf_ControllerMode-Indication()` has an invalid value, `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module, when `CanIf_ControllerModeIndication()` is called.⌋

[SWS_CANIF_00702] ⌈If `CanIf` was not initialized before calling `CanIf_ControllerModeIndication()`, `CanIf` shall not execute state transition notification, when `CanIf_ControllerModeIndication()` is called.⌋

Note: The call context of `CanIf_ControllerModeIndication()` is either on interrupt level (interrupt mode) or on task level (polling mode).

8.3.42 CanIf_TrcvModeIndication

[SWS_CANIF_00764] Definition of callback function CanIf_TrcvModeIndication ⌈

Service Name	CanIf_TrcvModeIndication	
Syntax	`void CanIf_TrcvModeIndication ( uint8 TransceiverId, CanTrcv_TrcvModeType TransceiverMode )`	
Service ID [hex]	0x22	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, which state has been transitioned.
	TransceiverMode	Mode to which the CAN transceiver transitioned
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates a transceiver state transition referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.	
Available via	CanIf.h	

⌋

Note: The callback service CanIf_TrcvModeIndication() is called by CanDrv and implemented in CanIf. It is called in case of a state transition notification of the CanDrv.

[SWS_CANIF_00706] ⌈If parameter TransceiverId of CanIf_TrcvMode-Indication() has an invalid value, CanIf shall report development error code CANIF_E_PARAM_TRCV to the Det_ReportError service of the DET module, when CanIf_TrcvModeIndication() is called.⌋

[SWS_CANIF_00708] ⌈If CanIf was not initialized before calling CanIf_TrcvMode-Indication(), CanIf shall not execute state transition notification, when CanIf_TrcvModeIndication() is called.⌋

Note: The call context of CanIf_TrcvModeIndication() is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00710] ⌈Configuration of CanIf_TrcvModeIndication(): ID of the CAN Transceiver is published inside the configuration description of CanIf via parameter CanIfTrcvId.⌋

[SWS_CANIF_00730] ⌈Configuration of CanIf_TrcvModeIndication(): If transceivers are not supported (CanIfTrcvDrvCfg is not configured), CanIf_Trcv-ModeIndication() shall not be provided by CanIf.⌋

8.3.43 CanIf_ControllerErrorStatePassive

[SWS_CANIF_91008] Definition of API function CanIf_ControllerErrorStatePassive ⌈

Service Name	CanIf_ControllerErrorStatePassive	
Syntax	`void CanIf_ControllerErrorStatePassive ( uint8 ControllerId, uint16 RxErrorCounter, uint16 TxErrorCounter )`	
Service ID [hex]	0x4f	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller.
	RxErrorCounter	Value of the Rx error counter
	TxErrorCounter	Value of the Tx error counter
Parameters (inout)	None	
Parameters (out)	None	
Return value	void	–

▽

△

Description	The function derives the ErrorCounterTreshold from RxErrorCounter/ TxErrorCounter values and reports it to the IdsM as security event SEV_CAN_ERRORSTATE_PASSIVE to the IdsM. It also prepares the context data for the respective security event.
Available via	CanIf.h

⌋

[SWS_CANIF_00919] ⌈If parameter `ControllerId` of `CanIf_ControllerError-StatePassive()` has an invalid value, the `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module when `CanIf_ControllerErrorStatePassive()` is called.⌋

8.3.44 CanIf_ErrorNotification

[SWS_CANIF_91009] Definition of API function CanIf_ErrorNotification ⌈

Service Name	CanIf_ErrorNotification	
Syntax	`void CanIf_ErrorNotification ( uint8 ControllerId, Can_ErrorType Can_ErrorType )`	
Service ID [hex]	0x50	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller.
	Can_ErrorType	Reported CAN error
Parameters (inout)	None	
Parameters (out)	None	
Return value	void	–
Description	The function shall derive the bus error source rx or tx from the parameter CanError and report the bus error as security event SEV_CAN_TX_ERROR_DETECTED or SEV_CAN_RX_ ERROR_DETECTED. It also prepares the context data for the respective security event.	
Available via	CanIf.h	

⌋

[SWS_CANIF_00920] ⌈If parameter `ControllerId` of `CanIf_Error-Notification()` has an invalid value, the `CanIf` shall report development error code `CANIF_E_PARAM_CONTROLLERID` to the `Det_ReportError` service of the DET module, when `CanIf_ErrorNotification()` is called.⌋

[SWS_CANIF_00921] ⌈If parameter `Can_ErrorType` of `CanIf_Error-Notification()` has an invalid value, the `CanIf` shall report development error code `CANIF_E_PARAM_CAN_ERROR` to the `Det_ReportError` service of the DET module, when `CanIf_ErrorNotification()` is called.⌋

8.3.45 CanIf_CanSecUpdateSecY

[SWS_CanIf_91019] Definition of API function CanIf_CanSecUpdateSecY ⌈

Service Name	CanIf_CanSecUpdateSecY	
Syntax	`Std_ReturnType CanIf_CanSecUpdateSecY ( uint8 ControllerId, uint8 SecYIdx, const Mka_CanSecConfigType* CANsecCfgPtr )`	
Service ID [hex]	0x57	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which the SecY shall be updated
	SecYIdx	SecY that shall be updated
	CANsecCfgPtr	Pointer to the structure to configure a CANsec Security Entity (SecY)
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been accepted `E_NOT_OK`: The request has not been accepted
Description	Requests the CAN Controller to update the SecY with the provided parameters and to configure the transmission Secure Channels as specified in the CAN Driver configuration.	
Available via	CanIf.h	

⌋

8.3.46 CanIf_CanSecAddRxSc

[SWS_CanIf_91020] Definition of API function CanIf_CanSecAddRxSc ⌈

Service Name	CanIf_CanSecAddRxSc	
Syntax	`Std_ReturnType CanIf_CanSecAddRxSc ( uint8 ControllerId, uint8 SecYIdx, uint16 Csci )`	
Service ID [hex]	0x58	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller for which a Rx Secure Channel shall be created
	SecYIdx	SecY for which a Rx Secure Channel shall be created
	Csci	CANsec Secure Channel Identifier for the CANsec's Reception Secure channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.

▽

△

Description	Requests the CAN Controller to add and initialize a Reception Secure Channel for the given CANsec Secure Channel Identifier
Available via	CanIf.h

⌋

8.3.47 CanIf_CanSecDeleteRxSc

[SWS_CanIf_91021] Definition of API function CanIf_CanSecDeleteRxSc ⌈

Service Name	CanIf_CanSecDeleteRxSc	

Syntax	`Std_ReturnType CanIf_CanSecDeleteRxSc (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci`
	`)`	

Service ID [hex]	0x59	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Rx Secure Channel shall be reset
	SecYIdx	SecYI for which a Rx Secure Channel shall be reset
	Csci	CANsec Secure Channel Identifier for the CANsec's Reception Secure channel
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Requests the CAN Controller to delete the Reception SecureChannel for the given CANsec Secure Channel Identifier.	
Available via	CanIf.h	

⌋

8.3.48 CanIf_CanSecAddTxSa

[SWS_CanIf_91022] Definition of API function CanIf_CanSecAddTxSa ⌈

Service Name	CanIf_CanSecAddTxSa

Syntax	`Std_ReturnType CanIf_CanSecAddTxSa (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An,`
	`  uint64 NextPn,`
	`  uint32 Ssci,`
	`  const Mka_SakKeyPtrType* KeysPtr,`
	`  boolean Active`
	`)`

Service ID [hex]	0x5a

▽

△

Sync/Async	Asynchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Tx SA shall be added
	SecYIdx	SecY for which a Tx SA shall be added
	Csci	CANsec Secure Channel Identifier for which the SA shall be added
	An	Association Number to use in the CANsec's transmission secure association
	NextPn	Next accepted Packet Number in the CANsec's transmission secure association
	Ssci	Short Secure Channel Identifiert used in the CANsec's transmission secure association
	KeysPtr	Pointer to the SAKs Key (and needed Key information) to use in the CANsec's transmission secure association
	Active	Boolean to enable/disable the CANsec's transmission secure association
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been accepted `E_NOT_OK`: The request has not been accepted
Description	Requests the CAN Controller to create a Transmission Secure Association in the Controller. The Short Secure Channel Identifier is included to support XPN configurations.	
Available via	CanIf.h	

⌋

8.3.49 CanIf_CanSecUpdateTxSa

[SWS_CanIf_91023] Definition of API function CanIf_CanSecUpdateTxSa ⌈

Service Name	CanIf_CanSecUpdateTxSa	

Syntax	`Std_ReturnType CanIf_CanSecUpdateTxSa (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An,`
	`  uint64 NextPn,`
	`  boolean Active`
	`)`	

Service ID [hex]	0x5b	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Tx SA shall be updated
	SecYIdx	SecY for which a Tx SA shall be updated
	Csci	CANsec Secure Channel Identifier for which the SA shall be updated
	An	Association Number to use in the CANsec's transmission secure association

▽

△

	NextPn	Next accepted Packet Number in the CANsec's transmission secure association
	Active	Boolean to enable/disable the CANsec's transmission secure association
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Requests the CAN Controller to update the Transmission Secure Association with the given Packet Number. The Active parameter is included to change the specified AN status.	
Available via	CanIf.h	

⌋

8.3.50 CanIf_CanSecDeleteTxSa

[SWS_CanIf_91024] Definition of API function CanIf_CanSecDeleteTxSa ⌈

Service Name	CanIf_CanSecDeleteTxSa	

Syntax	`Std_ReturnType CanIf_CanSecDeleteTxSa (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An`
	`)`	

Service ID [hex]	0x5c	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Tx SA shall be deleted
	SecYIdx	SecY for which a Tx SA shall be deleted
	Csci	CANsec Secure Channel Identifier for which the SA shall be deleted
	An	Association Number to use in the CANsec's transmission secure association
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Request the CAN Controller to remove the Transmission Secure Association identified by the provided Association Number.	
Available via	CanIf.h	

⌋

8.3.51 CanIf_CanSecAddRxSa

[SWS_CanIf_91025] Definition of API function CanIf_CanSecAddRxSa ⌈

Service Name	CanIf_CanSecAddRxSa	

Syntax	`Std_ReturnType CanIf_CanSecAddRxSa (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An,`
	`  uint64 LowestPn,`
	`  uint32 Ssci,`
	`  const Mka_SakKeyPtrType* KeysPtr,`
	`  boolean Active`
	`)`	

Service ID [hex]	0x5d	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Rx SA shall be added
	SecYIdx	SecY for which a Rx SA shall be added
	Csci	CANsec Secure Channel Identifier for which the SA shall be added
	An	Association Number to use in the CANsec's reception secure association
	LowestPn	Lowest accepted Packet Number in the CANsec's reception secure association
	Ssci	Short Secure Channel Identifiert used in the CANsec's reception secure association
	KeysPtr	Pointer to the SAKs Key (and needed Key information) to use in the CANsec's reception secure association
	Active	Boolean to enable/disable the CANsec's reception secure association
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been accepted `E_NOT_OK`: The request has not been accepted
Description	Requests the CAN Controller to create a Reception Secure Association in the Controller. The Short Secure Channel Identifier is included to support XPN configurations.	
Available via	CanIf.h	

⌋

8.3.52 CanIf_CanSecUpdateRxSa

[SWS_CanIf_91026] Definition of API function CanIf_CanSecUpdateRxSa ⌈

Service Name	CanIf_CanSecUpdateRxSa	

Syntax	`Std_ReturnType CanIf_CanSecUpdateRxSa (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An,`
	`  uint64 LowestPn,`
	`  boolean Active`
	`)`	

Service ID [hex]	0x5e	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Rx SA shall be updated
	SecYIdx	SecY for which a Rx SA shall be updated
	Csci	CANsec Secure Channel Identifier for which the SA shall be updated
	An	Association Number to use in the CANsec's reception secure association
	LowestPn	Lowest accepted Packet Number in the CANsec's reception secure association
	Active	Boolean to enable/disable the CANsec's reception secure association
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Requests the CAN Controller to update the Reception Secure Association with the given Packet Number. The Active parameter is included to change the specified AN status.	
Available via	CanIf.h	

⌋

8.3.53 CanIf_CanSecDeleteRxSa

[SWS_CanIf_91027] Definition of API function CanIf_CanSecDeleteRxSa ⌈

Service Name	CanIf_CanSecDeleteRxSa	

Syntax	`Std_ReturnType CanIf_CanSecDeleteRxSa (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An`
	`)`	

Service ID [hex]	0x5f	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Rx SA shall be deleted

▽

△

	SecYIdx	SecY for which a Rx SA shall be deleted
	Csci	CANsec Secure Channel Identifier for which the SA shall be deleted
	An	Association Number to use in the CANsec's reception secure association
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Request the CAN Controller to remove the Reception Secure Association identified by the provided Association Number.	
Available via	CanIf.h	

⌋

8.3.54 CanIf_CanSecGetTxSaNextPn

[SWS_CanIf_91028] Definition of API function CanIf_CanSecGetTxSaNextPn ⌈

Service Name	CanIf_CanSecGetTxSaNextPn	

Syntax	`Std_ReturnType CanIf_CanSecGetTxSaNextPn (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  uint8 An,`
	`  uint64* NextPnPtr`
	`)`	

Service ID [hex]	0x60	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller from which the NextPn shall be fetched
	SecYIdx	SecY for which the next PN shall be fetched
	Csci	CANsec Secure Channel Identifier for which the next PN shall be fetched
	An	Association Number to use in the CANsec's reception secure association
Parameters (inout)	None	
Parameters (out)	NextPnPtr	Pointer to the Next Packet Number read out from the CANsec Entity (SecY)
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Request the CAN Controller to return the Packet Number that is used for the next packet in the given Transmission Secure Association.	
Available via	CanIf.h	

⌋

8.3.55 CanIf_CanSecGetCanSecSecYStatistics

[SWS_CanIf_91029] Definition of API function CanIf_CanSecGetCanSecSecYStatistics ⌈

Service Name	CanIf_CanSecGetCanSecSecYStatistics	

Syntax	`Std_ReturnType CanIf_CanSecGetCanSecSecYStatistics (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  Mka_Stats_Tx_SecYType* CanSecTxSecYStatsPtr,`
	`  Mka_Stats_Rx_SecYType* CanSecRxSecYStatsPtr`
	`)`	

Service ID [hex]	0x76	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller from which CANsec statistics shall be fetched
	SecYIdx	SecY for which the statistics shall be fetched
Parameters (inout)	None	
Parameters (out)	CanSecTxSecYStatsPtr	Pointer to a structure including a set of SecY statistics by transmission
	CanSecRxSecYStatsPtr	Pointer to a structure including a set of SecY statistics by reception
Return value	Std_ReturnType	`E_OK`: The request has been accecpted `E_NOT_OK`: The request has not been accepted
Description	Request the CAN Controller to provide CANsec statistics for a given SecY.	
Available via	CanIf.h	

⌋

8.3.56 CanIf_CanSecGetCanSecTxScStatistics

[SWS_CanIf_91030] Definition of API function CanIf_CanSecGetCanSecTxScStatistics ⌈

Service Name	CanIf_CanSecGetCanSecTxScStatistics	

Syntax	`Std_ReturnType CanIf_CanSecGetCanSecTxScStatistics (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  Mka_Stats_Tx_ScType* CanSecTxScStatsPtr`
	`)`	

Service ID [hex]	0x6e	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller from which CANsec statistics shall be fetched
	SecYIdx	SecY for which the statistics shall be fetched
	Csci	CANsec Secure Channel Identifier of a transmission Secure Channel for which the statistics shall be fetched
Parameters (inout)	None	

▽

△

Parameters (out)	CanSecTxScStatsPtr	Pointer to a structure including the CANsec statistics of a CANsec transmission Secure Channel
Return value	Std_ReturnType	`E_OK`: The request has been accecpted `E_NOT_OK`: The request has not been accepted
Description	Request the CAN Controller to provide CANsec statistics for a given transmission Secure Channel.	
Available via	CanIf.h	

⌋

8.3.57 CanIf_CanSecGetCanSecRxScStatistics

[SWS_CanIf_91031] Definition of API function CanIf_CanSecGetCanSecRxSc Statistics ⌈

Service Name	CanIf_CanSecGetCanSecRxScStatistics	

Syntax	`Std_ReturnType CanIf_CanSecGetCanSecRxScStatistics (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  Mka_Stats_Rx_ScType* CanSecRxScStatsPtr`
	`)`	

Service ID [hex]	0x6f	
Sync/Async	Asynchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller from which CANsec statistics shall be fetched
	SecYIdx	SecY for which the statistics shall be fetched
	Csci	CANsec Secure Channel Identifier of a reception Secure Channel for which the statistics shall be fetched
Parameters (inout)	None	
Parameters (out)	CanSecRxScStatsPtr	Pointer to a structure including the CANsec statistics of a CANsec reception Secure Channel
Return value	Std_ReturnType	`E_OK`: The request has been accecpted `E_NOT_OK`: The request has not been accepted
Description	Request the CAN Controller to provide CANsec statistics for a given reception Secure Channel.	
Available via	CanIf.h	

⌋

8.3.58 CanIf_CanSecSetControlledPortEnabled

[SWS_CanIf_91032] Definition of API function CanIf_CanSecSetControlledPortEnabled ⌈

Service Name	CanIf_CanSecSetControlledPortEnabled	
Syntax	`Std_ReturnType CanIf_CanSecSetControlledPortEnabled ( uint8 ControllerId, uint8 SecYIdx, boolean ControlledPortEnabled )`	
Service ID [hex]	0x62	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which the Controlled Port shall be enabled
	SecYIdx	SecY from which the Controlled Port shall be enabled
	ControlledPortEnabled	Boolean to activate the Controlled Port of the SecY
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The request has been processed. `E_NOT_OK`: The request has not been processed.
Description	Requests to set the Controlled Port enabled parameter of a SecY.	
Available via	CanIf.h	

⌋

8.3.59 CanIf_CanSecUpdateSecYNotification

[SWS_CanIf_91033] Definition of callback function CanIf_CanSecUpdateSecYNotification ⌈

Service Name	CanIf_CanSecUpdateSecYNotification	
Syntax	`void CanIf_CanSecUpdateSecYNotification ( uint8 ControllerId, uint8 SecYIdx, Std_ReturnType Result )`	
Service ID [hex]	0x64	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which the SecY has been updated
	SecYIdx	SecY that has been updated
	Result	`E_OK`: Can_CanSecUpdateSecY has finished and SecY is updated with the provided parameters of Can_CanSecUpdate SecY `E_NOT_OK`: SecY has not been updated with the provided parameters of Can_CanSecUpdateSecY
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	None
Description	Callback to notify that Can_CanSecUpdateSecY finished.
Available via	CanIf.h

⌋

8.3.60 CanIf_CanSecAddTxSaNotification

[SWS_CanIf_91034] Definition of callback function CanIf_CanSecAddTxSaNotification ⌈

Service Name	CanIf_CanSecAddTxSaNotification	
Syntax	`void CanIf_CanSecAddTxSaNotification ( uint8 ControllerId, uint8 SecYIdx, uint16 Csci, uint8 An, Std_ReturnType Result )`	
Service ID [hex]	0x65	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Tx SA has been added
	SecYIdx	SecY for which a Tx SA has been added
	Csci	CANsec Secure Channel Identifier for which the SA has been added
	An	Association Number used in the CANsec's transmission secure association
	Result	`E_OK`: Can_CanSecAddTxSa has finished and Transmission Secure Association is created `E_NOT_OK`: The Transmission Secure Association is not created through Can_CanSecAddTxSa.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback to notify that Can_CanSecAddTxSa has finished.	
Available via	CanIf.h	

⌋

8.3.61 CanIf_CanSecAddRxSaNotification

[SWS_CanIf_91035] Definition of callback function CanIf_CanSecAddRxSaNotification ⌈

Service Name	CanIf_CanSecAddRxSaNotification	
Syntax	`void CanIf_CanSecAddRxSaNotification ( uint8 ControllerId, uint8 SecYIdx, uint16 Csci, uint8 An, Std_ReturnType Result )`	
Service ID [hex]	0x66	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which a Rx SA has been added
	SecYIdx	SecY for which a Rx SA has been added
	Csci	CANsec Secure Channel Identifier for which the SA has been added
	An	Association Number used in the CANsec's reception secure association
	Result	`E_OK`: Can_CanSecAddRxSa has finished and Reception Secure Association is created `E_NOT_OK`: The Reception Secure Association is not created through Can_CanSecAddRxSa.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback to notify that Can_CanSecAddRxSa has finished.	
Available via	CanIf.h	

⌋

8.3.62 CanIf_CanSecGetCanSecSecYStatisticsNotification

[SWS_CanIf_91036] Definition of callback function CanIf_CanSecGetCanSecSecYStatisticsNotification ⌈

Service Name	CanIf_CanSecGetCanSecSecYStatisticsNotification	
Syntax	`void CanIf_CanSecGetCanSecSecYStatisticsNotification ( uint8 ControllerId, uint8 SecYIdx, Std_ReturnType Result )`	
Service ID [hex]	0x78	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which CANsec statistics have been requested
	SecYIdx	SecY for which the statistics have been requested

▽

△

	Result	E_OK: CanSecStatistics have been received E_NOT_OK: CanSecStatistics have not been received
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback to notify that Can_CanSecGetCanSecSecYStatistics has finished and provides the requested statistics.	
Available via	CanIf.h	

⌋

8.3.63 CanIf_CanSecGetCanSecTxScStatisticsNotification

[SWS_CanIf_91037] Definition of callback function CanIf_CanSecGetCanSecTxScStatisticsNotification ⌈

Service Name	CanIf_CanSecGetCanSecTxScStatisticsNotification	

Syntax	`void CanIf_CanSecGetCanSecTxScStatisticsNotification (`
	`  uint8 ControllerId,`
	`  uint8 SecYIdx,`
	`  uint16 Csci,`
	`  Std_ReturnType Result`
	`)`	

Service ID [hex]	0x71	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which CANsec statistics have been requested
	SecYIdx	SecY for which the statistics have been requested
	Csci	CANsec Secure Channel Identifier of a transmission Secure Channel for which the statistics have been requested
	Result	E_OK: CanSecStatistics have been received E_NOT_OK: CanSecStatistics have not been received
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback to notify that Can_CanSecGetCanSecTxScStatistics has finished and provides the requested statistics.	
Available via	CanIf.h	

⌋

8.3.64 CanIf_CanSecGetCanSecRxScStatisticsNotification

[SWS_CanIf_91038] Definition of callback function CanIf_CanSecGetCanSecRxScStatisticsNotification ⌈

Service Name	CanIf_CanSecGetCanSecRxScStatisticsNotification	
Syntax	`void CanIf_CanSecGetCanSecRxScStatisticsNotification ( uint8 ControllerId, uint8 SecYIdx, uint16 Csci, Std_ReturnType Result )`	
Service ID [hex]	0x72	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different ControllerId, Non reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN Controller for which CANsec statistics have been requested
	SecYIdx	SecY for which the statistics have been requested
	Csci	CANsec Secure Channel Identifier of a reception Secure Channel for which the statistics have been requested
	Result	`E_OK`: CanSecStatistics have been received `E_NOT_OK`: CanSecStatistics have not been received
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback to notify that Can_CanSecGetCanSecRxScStatistics has finished and provides the requested statistics.	
Available via	CanIf.h	

⌋

8.4 Scheduled functions

Note: `CanIf` does not have scheduled functions or needs some.

8.5 Expected interfaces

In this chapter all interfaces required from other modules are listed.

8.5.1 Mandatory interfaces

Note: This section defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_CANIF_00040] Definition of mandatory interfaces required by module CanIf ⌈

API Function	Header File	Description
Can_GetControllerErrorState	Can.h	This service obtains the error state of the CAN controller.
Can_GetControllerRxErrorCounter	Can.h	Returns the Rx error counter for a CAN controller. This value might not be available for all CAN controllers, in which case E_NOT_OK would be returned. Please note that the value of the counter might not be correct at the moment the API returns it, because the Rx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state.
Can_GetControllerTxErrorCounter	Can.h	Returns the Tx error counter for a CAN controller. This value might not be available for all CAN controllers, in which case E_NOT_OK would be returned. Please note that the value of the counter might not be correct at the moment the API returns it, because the Tx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state.
Can_SetControllerMode	Can.h	This function performs software triggered state transitions of the CAN controller State machine.
Can_Write	Can.h	This function is called by CanIf to pass a CAN message to CanDrv for transmission.
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
SchM_Enter_CanIf_<ExclusiveArea>	SchM_<Mip>.h	Invokes the SchM_Enter function to enter a module local exclusive area.
SchM_Exit_CanIf_<ExclusiveArea>	SchM_<Mip>.h	Invokes the SchM_Exit function to exit an exclusive area.

⌋

8.5.2 Optional interfaces

This section defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_CANIF_00294] Definition of optional interfaces requested by module CanIf ⌈

API Function	Header File	Description
Can_CanSecAddRxSa	Can.h	Requests the CAN Controller to create a Reception Secure Association in the Controller. The Short Secure Channel Identifier is included to support XPN configurations.
Can_CanSecAddRxSc	Can.h	Requests the CAN Controller to add and initialize a Reception Secure Channel for the given CANsec Secure Channel Identifier

▽

△

API Function	*Header File*	*Description*
Can_CanSecAddTxSa	Can.h	Requests the CAN Controller to create a Transmission Secure Association in the Controller. The Short Secure Channel Identifier is included to support XPN configurations.
Can_CanSecDeleteRxSa	Can.h	Request the CAN Controller to remove the Reception Secure Association identified by the provided Association Number.
Can_CanSecDeleteRxSc	Can.h	Requests the CAN Controller to delete the Reception SecureChannel for the given CANsec Secure Channel Identifier.
Can_CanSecDeleteTxSa	Can.h	Request the CAN Controller to remove the Transmission Secure Association identified by the provided Association Number.
Can_CanSecGetCanSecRxSc Statistics	Can.h	Request the CAN Controller to provide CANsec statistics for a given reception Secure Channel.
Can_CanSecGetCanSecSec YStatistics	Can.h	Request the CAN Controller to provide CANsec statistics for a given SecY.
Can_CanSecGetCanSecTxSc Statistics	Can.h	Request the CAN Controller to provide CANsec statistics for a given transmission Secure Channel.
Can_CanSecGetTxSaNextPn	Can.h	Request the CAN Controller to return the Packet Number that is used for the next packet in the given Transmission Secure Association.
Can_CanSecSetControlledPort Enabled	Can.h	Requests to set the Controlled Port enabled parameter of a SecY.
Can_CanSecUpdateRxSa	Can.h	Requests the CAN Controller to update the Reception Secure Association with the given Packet Number. The Active parameter is included to change the specified AN status.
Can_CanSecUpdateSecY	Can.h	Requests the CAN Controller to update the SecY with the provided parameters and to configure the transmission Secure Channels as specified in the CAN Driver configuration.
Can_CanSecUpdateTxSa	Can.h	Requests the CAN Controller to update the Transmission Secure Association with the given Packet Number. The Active parameter is included to change the specified AN status.
Can_CheckWakeup	Can.h	This function checks if a wakeup has occurred for the given controller.
Can_SetBaudrate	Can.h	This service shall set the baud rate configuration of the CAN controller. Depending on necessary baud rate modifications the controller might have to reset.
CanSM_CheckTransceiverWakeFlag Indication	CanSM_CanIf.h	This callback function indicates the CanIf_Check TrcvWakeFlag API process end for the notified CAN Transceiver.
CanSM_ClearTrcvWufFlagIndication	CanSM_CanIf.h	This callback function shall indicate the CanIf_Clear TrcvWufFlag API process end for the notified CAN Transceiver.
CanSM_ConfirmCtrlPnAvailability (draft)	CanSM_CanIf.h	This callback function indicates that the controller is running in PN communication mode. **Tags:** atp.Status=draft
CanSM_ConfirmPnAvailability	CanSM_CanIf.h	This callback function indicates that the transceiver is running in PN communication mode.

▽

△

API Function	*Header File*	*Description*
CanSM_ControllerBusOff	CanSM_CanIf.h	This callback function notifies the CanSM about a bus-off event on a certain CAN controller, which needs to be considered with the specified bus-off recovery handling for the impacted CAN network.
CanSM_ControllerModeIndication	CanSM_CanIf.h	This callback shall notify the CanSM module about a CAN controller mode change.
CanSM_TransceiverModeIndication	CanSM_CanIf.h	This callback shall notify the CanSM module about a CAN transceiver mode change.
CanTrcv_CheckWakeFlag	CanTrcv.h	Requests to check the status of the wakeup flag from the transceiver hardware.
CanTrcv_CheckWakeup	CanTrcv.h	Service is called by underlying CANIF in case a wake up interrupt is detected.
CanTrcv_GetBusWuReason	CanTrcv.h	Gets the wakeup reason for the Transceiver and returns it in parameter Reason.
CanTrcv_GetOpMode	CanTrcv.h	Gets the mode of the Transceiver and returns it in OpMode.
CanTrcv_SetOpMode	CanTrcv.h	Sets the mode of the Transceiver to the value Op Mode.
CanTrcv_SetWakeupMode	CanTrcv.h	Enables, disables or clears wake-up events of the Transceiver according to TrcvWakeupMode.
CanXL_Write	CanXL.h	This function is called by CanIf to pass a CAN XL message to the CAN XL driver for transmission. It provides the CAN XL specific parameters besides the hardware handle and actual PDU information.
CanXL_XLReleaseRxBuffer	CanXL.h	Indication from CanIf to release the CAN XL reception buffer.
CanXLTrcv_ReportErrorState	CanXLTrcv.h	Reports each change of the CAN error state.
Det_ReportError	Det.h	Service to report development errors.
EcuM_ValidateWakeupEvent	EcuM.h	After wakeup, the ECU State Manager will stop the process during the WAKEUP VALIDATION state/ sequence to wait for validation of the wakeup event.This API service is used to indicate to the ECU Manager module that the wakeup events indicated in the sources parameter have been validated.
IdsM_ReportSecurityEvent	IdsM.h	This API is the interface to report security events to the IdsM.
LSduR_CanIfRxIndication (draft)	LSduR_CanIf.h	Indication of a received PDU from a lower layer communication interface module.
LSduR_CanIfTriggerTransmit (draft)	LSduR_CanIf.h	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->Sdu Length. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->Sdu Length. If not, it returns E_NOT_OK without changing PduInfoPtr.
LSduR_CanIfTxConfirmation (draft)	LSduR_CanIf.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
Mirror_ReportCanFrame	Mirror.h	Reports a received or transmitted CAN frame. All received CAN frames that pass the hardware acceptance filter are reported, independent of the software filter configuration. Transmitted CAN frames are reported when the transmission is confirmed.

▽

△

API Function	*Header File*	*Description*
Mka_CanSecAddRxSaNotification	Mka.h	Callback to notify that CanIf_CanSecAddRxSa has finished. Note: This API is preliminary and must be implemented by integration code to complete the CANsec control plane. Functional specification will be added once the CANsec key agreement specification is released by CiA.
Mka_CanSecAddTxSaNotification	Mka.h	Callback to notify that CanIf_CanSecAddTxSa has finished. Note: This API is preliminary and must be implemented by integration code to complete the CANsec control plane. Functional specification will be added once the CANsec key agreement specification is released by CiA.
Mka_CanSecUpdateSecYNotification	Mka.h	Callback to to notify that CanIf_CanSecUpdateSecY finished. Note: This API is preliminary and must be implemented by integration code to complete the CANsec control plane. Functional specification will be added once the CANsec key agreement specification is released by CiA.
Mka_GetCanSecRxScStatistics Notification	Mka.h	Callback to notify that CanIf_CanSecGetCanSecRx ScStatistics has finished and provides the requested statistics. Note: This API is preliminary and must be implemented by integration code to complete the CANsec control plane. Functional specification will be added once the CANsec key agreement specification is released by CiA.
Mka_GetCanSecSecYStatistics Notification	Mka.h	Callback notify that Can_GetCanSecSecYStatistics has finished and provides the requested statistics. Note: This API is preliminary and must be implemented by integration code to complete the CANsec control plane. Functional specification will be added once the CANsec key agreement specification is released by CiA.
Mka_GetCanSecTxScStatistics Notification	Mka.h	Callback to notify that CanIf_CanSecGetCanSecTx ScStatistics has finished and provides the requested statistics. Note: This API is preliminary and must be implemented by integration code to complete the CANsec control plane. Functional specification will be added once the CANsec key agreement specification is released by CiA.

⌋

8.5.3 Configurable interfaces

In this section all interfaces are listed, where the target function of any upper layer to be called has to be set up by configuration. These callback services are specified and implemented in the upper communication modules, which use `CanIf` according to the AUTOSAR BSW architecture. The specific callback notification is specified in the corresponding SWS document (see Chapter 3 “Related documentation”).

As far the interface name is not specified to be mandatory, no callback is performed, if no API name is configured. This section describes only the content of notification of the callback, the call context inside CanIf and exact time by the call event.

<User_NotificationName> - This condition is applied for such interface services which will be implemented in the upper layer and called by CanIf. This condition displays the symbolic name of the functional group in a callback service in the corresponding upper layer module. Each upper layer module can define no, one or several callback services for the same functionality (i.e. *transmit confirmation*). The dispatch is ensured by the L-SDU ID.

The upper layer module provides the *Service ID* of the following functions.

8.5.3.1 <User_ValidateWakeupEvent>

[SWS_CANIF_00532] Definition of configurable interface <User_ValidateWakeupEvent> ⌈

Service Name	<User_ValidateWakeupEvent>	
Syntax	`void <User_ValidateWakeupEvent> ( EcuM_WakeupSourceType sources )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant (defined within providing upper layer module)	
Parameters (in)	sources	Validated CAN wakeup events. Every CAN controller or CAN transceiver can be a separate wakeup source.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates if a wake up event initiated from the wake up source (CAN controller or transceiver) after a former request to the CAN Driver or CAN Transceiver Driver module is valid.	
Available via	Configuration parameter CanIfPublicCddHeaderFile	

⌋

Note: This callback service is mainly implemented in and used by the *ECU State Manager* module (see [12, Specification of ECU State Manager]).

Note: The CanIf calls this callback service. It is implemented by the configured upper layer module. It is called only during the call of CanIf_CheckValidation() if a first CAN L-PDU reception event after a wake up event has been occurred at the corresponding CAN Controller.

Note: The call context of <User_ValidateWakeupEvent>() is either on interrupt level (interrupt mode) or on task level (polling mode).

Note: The callback service <User_ValidateWakeupEvent>() is in general reentrant for multiple CAN Controller usage, but not for the same CAN Controller

[SWS_CANIF_00659] ⌈Configuration of <User_ValidateWakeupEvent>(): If no validation is needed, this API can be omitted by disabling CanIfPublicWakeupCheckValidSupport.⌋

[SWS_CANIF_00456] ⌈Configuration of <User_ValidateWakeupEvent>(): The upper layer module which provides this callback service has to be configured by CanIfDispatchUserValidateWakeupEventUL, but:

- If no upper layer modules are configured for wake up notification using <User_ValidateWakeupEvent>(), no wake up notification needs to be configured. CanIfDispatchUserValidateWakeupEventUL needs not to be configured.
- If wake up is not supported (CanIfCtrlWakeupSupport and CanIfTrcvWakeupSupport equal FALSE, CanIfDispatchUserValidateWakeupEventUL is not configurable.

⌋

[SWS_CANIF_00563] ⌈Configuration of <User_ValidateWakeupEvent>(): If CanIfDispatchUserValidateWakeupEventUL is set to ECUM, CanIfDispatchUserValidateWakeupEventName must be EcuM_ValidateWakeupEvent.⌋

[SWS_CANIF_00564] ⌈Configuration of <User_ValidateWakeupEvent>(): If CanIfDispatchUserValidateWakeupEventUL is set to CDD the name of the API has to be configured via parameter CanIfDispatchUserValidateWakeupEventName.⌋

8.5.3.2 <User_ControllerBusOff>

[SWS_CANIF_00014] Definition of configurable interface <User_ControllerBusOff> ⌈

Service Name	<User_ControllerBusOff>	
Syntax	`void <User_ControllerBusOff> ( uint8 ControllerId )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant (defined within providing upper layer module)	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, at which a BusOff occurred.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates a bus-off event to the corresponding upper layer module (mainly the CAN State Manager module).	
Available via	Configuration parameter CanIfPublicCddHeaderFile	

⌋

Note: This callback service is mainly implemented in and used by CanSm (see [3, Specification of CAN State Manager]).

Note: This callback service is called by CanIf and implemented by the configured upper layer module. It is called in case of a *BusOff notification* via CanIf_ControllerBusOff() of the CanDrv. The delivered parameter ControllerId of the service CanIf_ControllerBusOff() is passed to the upper layer module.

Note: The call context of <User_ControllerBusOff>() is either on interrupt level (*interrupt mode*) or on task level (*polling mode*).

Note: The callback service <User_ControllerBusOff>() is in general re-entrant for multiple CAN Controller usage, but not for the same CAN Controller.

Note: Before re-initialization/restart during *BusOff recovery* is executed <User_ControllerBusOff>() is performed only once in case of multiple *BusOff events* at CAN Controller.

Configuration of <User_ControllerBusOff>()

[SWS_CANIF_00450] ⌈Configuration of <User_ControllerBusOff>(): The upper layer module which provides this callback service has to be configured by CanIfDispatchUserCtrlBusOffUL.⌋

[SWS_CANIF_00558] ⌈Configuration of <User_ControllerBusOff>(): The name of the API <User_ControllerBusOff>() which will be called by CanIf shall be configured for CanIf by parameter CanIfDispatchUserCtrlBusOffName.⌋

[SWS_CANIF_00524] ⌈Configuration of <User_ControllerBusOff>(): At least one upper layer module and hence an API of <User_ControllerBusOff>() has mandatorily to be configured, which CanIf can call in case of an occurred call of CanIf_ControllerBusOff().⌋

[SWS_CANIF_00559] ⌈Configuration of <User_ControllerBusOff>(): If CanIfDispatchUserCtrlBusOffUL is set to CAN_SM, CanIfDispatchUserCtrlBusOffName must be CanSM_ControllerBusOff.⌋

[SWS_CANIF_00560] ⌈Configuration of <User_ControllerBusOff>(): If CanIfDispatchUserCtrlBusOffUL is set to CDD the name of the API has to be configured via parameter CanIfDispatchUserCtrlBusOffName.⌋

8.5.3.3 <User_ConfirmPnAvailability>

[SWS_CANIF_00821] Definition of configurable interface <User_ConfirmPnAvailability> ⌈

Service Name	<User_ConfirmPnAvailability>	
Syntax	`void <User_ConfirmPnAvailability> ( uint8 TransceiverId )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant (defined within providing upper layer module)	
Parameters (in)	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, which was checked for PN availability.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates that the CAN transceiver is running in PN communication mode.	
Available via	Configuration parameter CanIfPublicCddHeaderFile	

⌋

Note: This callback service is mainly implemented in and used by CanSm (see [3, Specification of CAN State Manager]).

Note: The call context of <User_ConfirmPnAvailability>() is either on interrupt level (interrupt mode) or on task level (polling mode).

Note: The callback service <User_ConfirmPnAvailability>() is in general reentrant for multiple CAN Controller usage, but not for the same CAN Controller

[SWS_CANIF_00823] ⌈Configuration of <User_ConfirmPnAvailability>(): The upper layer module, which is called (see [SWS_CANIF_00753]), has to be configurable by CanIfDispatchUserConfirmPnAvailabilityUL if CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True.⌋

[SWS_CANIF_00824] ⌈Configuration of <User_ConfirmPnAvailability>(): The name of <User_ConfirmPnAvailability>() shall be configurable by CanIfDispatchUserConfirmPnAvailabilityName if CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True.⌋

[SWS_CANIF_00825] ⌈Configuration of <User_ConfirmPnAvailability>(): It shall be configurable by CanIfPublicPnSupport and CanIfPublicTrcvPnEnable, if CanIf supports this service (False: not supported, True: supported)⌋

[SWS_CANIF_00826] ⌈Configuration of <User_ConfirmPnAvailability>(): If CanIfDispatchUserConfirmPnAvailabilityUL is set to CAN_SM, CanIfDispatchUserConfirmPnAvailabilityName must be CanSM_ConfirmPnAvailability.⌋

[SWS_CANIF_00827] ⌈Configuration of <User_ConfirmPnAvailability>(): If CanIfDispatchUserConfirmPnAvailabilityUL is set to CDD, the name of the

service has to be configurable via parameter CanIfDispatchUserConfirmPnAvailabilityName.⌋

8.5.3.4 <User_ConfirmCtrlPnAvailability>

[SWS_CANIF_91017] Definition of configurable interface <User_ConfirmCtrlPnAvailability>

Status: DRAFT

⌈

Service Name	<User_ConfirmCtrlPnAvailability> (draft)	

Syntax	`void <User_ConfirmCtrlPnAvailability> (`
	`  uint8 ControllerId`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant (defined within providing upper layer module)	
Parameters (in)	ControllerId	Abstract CanIf ControllerId, which is assigned to a CAN controller, which was checked for PN availability.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates that the CAN controller is running in PN communication mode. **Tags:** atp.Status=draft	
Available via	configurable	

⌋

Note: This callback service is mainly implemented in and used by CanSm (see [3, Specification of CAN State Manager]).

Note: The call context of <User_ConfirmCtrlPnAvailability>() is either on interrupt level (interrupt mode) or on task level (polling mode).

[SWS_CANIF_00958] ⌈Configuration of <User_ConfirmCtrlPnAvailability>(): The upper layer module, which is called (see [SWS_CANIF_00963]), has to be configurable by CanIfDispatchUserConfirmCtrlPnAvailabilityUL if CanIfPublicPnSupport equals True and CanIfPublicCtrlPnEnable equals True.⌋

[SWS_CANIF_00959] ⌈Configuration of <User_ConfirmCtrlPnAvailability>(): The name of <User_ConfirmCtrlPnAvailability>() shall be configurable by CanIfDispatchUserConfirmCtrlPnAvailabilityName if CanIfPublicPnSupport equals True and CanIfPublicCtrlPnEnable equals True.⌋

[SWS_CANIF_00960] ⌈Configuration of <User_ConfirmCtrlPnAvailability>(): It shall be configurable by CanIfPublicPnSupport and

CanIfPublicCtrlPnEnable, if CanIf supports this service (False: not supported, True: supported)⌋

[SWS_CANIF_00961] ⌈Configuration of <User_ConfirmCtrlPnAvailability>(): If CanIfDispatchUserConfirmCtrlPnAvailabilityUL is set to CAN_SM, CanIfDispatchUserConfirmCtrlPnAvailabilityName must be CanSM_ConfirmCtrlPnAvailability.⌋

[SWS_CANIF_00962] ⌈Configuration of <User_ConfirmCtrlPnAvailability>(): If CanIfDispatchUserConfirmCtrlPnAvailabilityUL is set to CDD, the name of the service has to be configurable via parameter CanIfDispatchUserConfirmCtrlPnAvailabilityName.⌋

8.5.3.5 <User_ClearTrcvWufFlagIndication>

[SWS_CANIF_00788] Definition of configurable interface <User_ClearTrcvWufFlagIndication> ⌈

Service Name	<User_ClearTrcvWufFlagIndication>	
Syntax	`void <User_ClearTrcvWufFlagIndication> ( uint8 TransceiverId )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId, for which this function was called.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates that the CAN transceiver has cleared the WufFlag. This function is called in CanIf_ClearTrcvWufFlagIndication.	
Available via	Configuration parameter CanIfPublicCddHeaderFile	

⌋

Note: This callback service is mainly implemented in and used by CanSm (see [3, Specification of CAN State Manager]).

Note: The call context of <User_ClearTrcvWufFlagIndication>() is either on interrupt level (interrupt mode) or on task level (polling mode).

Note: The callback service <User_ClearTrcvWufFlagIndication>() is in general re-entrant for multiple CAN Controller usage, but not for the same CAN Controller

[SWS_CANIF_00794] ⌈Configuration of <User_ClearTrcvWufFlagIndication>(): The upper layer module, which is called (see [SWS_CANIF_00757]), has to be configurable by CanIfDispatchUserClearTrcvWufFlagIndicationUL if CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True.⌋

[SWS_CANIF_00795] ⌈Configuration of <User_ClearTrcvWufFlagIndication>(): The name of <User_ClearTrcvWufFlagIndication>() shall be configurable by CanIfDispatchUserClearTrcvWufFlagIndicationName if CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True.⌋

[SWS_CANIF_00796] ⌈Configuration of <User_ClearTrcvWufFlagIndication>(): It shall be configurable by CanIfPublicPnSupport and CanIfPublicTrcvPnEnable, if CanIf supports this service (False: not supported, True: supported)⌋

[SWS_CANIF_00797] ⌈Configuration of <User_ClearTrcvWufFlagIndication>(): If CanIfDispatchUserClearTrcvWufFlagIndicationUL is set to CAN_SM, CanIfDispatchUserClearTrcvWufFlagIndicationName must be CanSM_ClearTrcvWufFlagIndication.⌋

[SWS_CANIF_00798] ⌈Configuration of <User_ClearTrcvWufFlagIndication>(): If CanIfDispatchUserClearTrcvWufFlagIndicationUL is set to CDD, the name of the service has to be configurable via parameter CanIfDispatchUserClearTrcvWufFlagIndicationName.⌋

8.5.3.6 <User_CheckTrcvWakeFlagIndication>

[SWS_CANIF_00814] Definition of configurable interface <User_CheckTrcvWakeFlagIndication> ⌈

Service Name	<User_CheckTrcvWakeFlagIndication>	
Syntax	`void <User_CheckTrcvWakeFlagIndication> ( uint8 TransceiverId )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId, for which this function was called.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates that the wake up flag in the CAN transceiver is set. This function is called in CanIf_CheckTrcvWakeFlagIndication.	
Available via	Configuration parameter CanIfPublicCddHeaderFile	

⌋

Note: This callback service is mainly implemented in and used by CanSm (see [3, Specification of CAN State Manager]).

Note: The call context of <User_CheckTrcvWakeFlagIndication>() is either on interrupt level (interrupt mode) or on task level (polling mode).

Note: The callback service <User_CheckTrcvWakeFlagIndication>() is in general re-entrant for multiple CAN Controller usage, but not for the same CAN Controller

[SWS_CANIF_00800] ⌈Configuration of <User_CheckTrcvWakeFlagIndication>(): The upper layer module, which is called (see [SWS_CANIF_00759]), has to be configurable by CanIfDispatchUserCheckTrcvWakeFlagIndicationUL if CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True.⌋

[SWS_CANIF_00801] ⌈Configuration of <User_CheckTrcvWakeFlagIndication>(): The name of <User_CheckTrcvWakeFlagIndication>() shall be configurable by CanIfDispatchUserCheckTrcvWakeFlagIndicationName if CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True.⌋

[SWS_CANIF_00802] ⌈Configuration of <User_CheckTrcvWakeFlagIndication>(): It shall be configurable by CanIfPublicPnSupport and CanIfPublicTrcvPnEnable, if CanIf supports this service (False: not supported, True: supported)⌋

[SWS_CANIF_00803] ⌈Configuration of <User_CheckTrcvWakeFlagIndication>(): If CanIfDispatchUserCheckTrcvWakeFlagIndicationUL is set to CAN_SM, CanIfDispatchUserCheckTrcvWakeFlagIndicationName must be CanSM_CheckTransceiverWakeFlagIndication.⌋

[SWS_CANIF_00804] ⌈Configuration of <User_CheckTrcvWakeFlagIndication>(): If CanIfDispatchUserCheckTrcvWakeFlagIndicationUL is set to CDD, the name of the service has to be configurable via parameter CanIfDispatchUserCheckTrcvWakeFlagIndicationName.⌋

8.5.3.7 <User_ControllerModeIndication>

[SWS_CANIF_00687] Definition of configurable interface <User_ControllerModeIndication> ⌈

Service Name	<User_ControllerModeIndication>	

Syntax	`void <User_ControllerModeIndication> (`
	`  uint8 ControllerId,`
	`  Can_ControllerStateType ControllerMode`
	`)`	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, at which a controller state transition occurred.
	ControllerMode	Notified CAN controller mode
Parameters (inout)	None	

▽

△

Parameters (out)	None
Return value	None
Description	This service indicates a CAN controller state transition to the corresponding upper layer module (mainly the CAN State Manager module).
Available via	Configuration parameter CanIfPublicCddHeaderFile

⌋

Note: The upper layer module provides the Service ID.

Note: This callback service is mainly implemented in and used by CanSm (see [3, Specification of CAN State Manager]).

Note: The CanIf calls this callback service. It is implemented by the configured upper layer module. It is called in case of a *state transition notification* via CanIf_ControllerModeIndication() of the CanDrv. The delivered parameter ControllerId of the service CanIf_ControllerModeIndication() is passed to the upper layer module. The delivered parameter ControllerMode of the service CanIf_ControllerModeIndication() is mapped to the appropriate parameter ControllerMode of <User_ControllerModeIndication>().

Note: For different upper layer users different service names shall be used.

Note: The call context of <User_ControllerModeIndication>() is on task level (polling mode).

Note: The callback service <User_ControllerModeIndication>() is in general re-entrant for multiple CAN Controller usage, but not for the same CAN Controller

[SWS_CANIF_00689] ⌈Configuration of <User_ControllerModeIndication>(): The upper layer module which provides this callback service has to be configured by CanIfDispatchUserCtrlModeIndicationUL.⌋

[SWS_CANIF_00690] ⌈Configuration of <User_ControllerModeIndication>(): The name of <User_ControllerModeIndication>() which is called by CanIf shall be configured for CanIf by parameter CanIfDispatchUserCtrlModeIndicationName. This is only necessary if *state transition notifications* are configured via CanIfDispatchUserCtrlModeIndicationUL.⌋

[SWS_CANIF_00691] ⌈Configuration of <User_ControllerModeIndication>(): If CanIfDispatchUserCtrlModeIndicationUL is set to CAN_SM, CanIfDispatchUserCtrlModeIndicationName must be CanSM_ControllerModeIndication.⌋

[SWS_CANIF_00972] ⌈Configuration of <User_ControllerModeIndication>(): If CanIfDispatchUserCtrlModeIndicationUL is set to MKA, CanIfDispatchUserCtrlModeIndicationName must be Mka_CanControllerModeIndication.⌋

[SWS_CANIF_00692] ⌈Configuration of `<User_ControllerModeIndication>()`: If `CanIfDispatchUserCtrlModeIndicationUL` is set to `CDD` the name of the function has to be configured via parameter `CanIfDispatchUserCtrlModeIndicationName`.⌋

8.5.3.8 <User_TrcvModeIndication>

[SWS_CANIF_00693] Definition of configurable interface <User_TrcvModeIndication> ⌈

Service Name	<User_TrcvModeIndication>	
Syntax	`void <User_TrcvModeIndication> ( uint8 TransceiverId, CanTrcv_TrcvModeType TransceiverMode )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TransceiverId	Abstracted CanIf TransceiverId which is assigned to a CAN transceiver, at which a transceiver state transition occurred.
	TransceiverMode	Notified CAN transceiver mode
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service indicates a CAN transceiver state transition to the corresponding upper layer module (mainly the CAN State Manager module).	
Available via	Configuration parameter CanIfPublicCddHeaderFile	

⌋

Note: The upper layer module provides the Service ID.

Note: This callback service is mainly implemented in and used by `CanSm` (see [3, Specification of CAN State Manager]).

Note: The `CanIf` calls this callback service. It is implemented by the configured upper layer module. It is called in case of a *state transition notification* via `CanIf_TrcvModeIndication()` of the `CanTrcv`. The delivered parameter `TransceiverId` of the service `CanIf_TrcvModeIndication()` is mapped (as configured) to the appropriate parameter `TransceiverId` which will be passed to the upper layer module. The delivered parameter `TransceiverMode` of the service `CanIf_TrcvModeIndication()` is mapped to the appropriate parameter `TransceiverMode` of `<User_TrcvModeIndication>()`.

Note: For different upper layer users different service names shall be used.

[SWS_CANIF_00694] ⌈Caveats of `<User_TrcvModeIndication>()`:

- The `CanTrcv` must be initialized after *Power ON*.
- The call context is either on task level (*polling mode*).

- This callback service is in general re-entrant for multiple CAN Transceiver usage, but not for the same CAN Transceiver.

⌋

[SWS_CANIF_00695] ⌈Configuration of <User_TrcvModeIndication>(): The upper layer module which provides this callback service has to be configured by CanIfDispatchUserTrcvModeIndicationUL, but:

- If no upper layer modules are configured for *transceiver mode indications* using <User_TrcvModeIndication>(), no *transceiver mode indication* needs to be configured. CanIfDispatchUserTrcvModeIndicationUL needs not to be configured.
- If transceivers are not supported (CanIfTrcvDrvCfg is not configured, CanIfDispatchUserTrcvModeIndicationUL is not configurable.

⌋

If no upper layer modules are configured for *state transition notifications* using <User_TrcvModeIndication>(), no *state transition notification* needs to be configured.

[SWS_CANIF_00696] ⌈Configuration of <User_TrcvModeIndication>(): The name of <User_TrcvModeIndication>() which will be called by CanIf shall be configured for CanIf by parameter CanIfDispatchUserTrcvModeIndicationName. This is only necessary if *state transition notifications* are configured via CanIfDispatchUserTrcvModeIndicationUL.⌋

[SWS_CANIF_00697] ⌈Configuration of <User_TrcvModeIndication>(): If CanIfDispatchUserTrcvModeIndicationUL is set to CAN_SM, CanIfDispatchUserTrcvModeIndicationName must be CanSM_TransceiverModeIndication.⌋

[SWS_CANIF_00698] ⌈Configuration of <User_TrcvModeIndication>(): If CanIfDispatchUserTrcvModeIndicationUL is set to CDD the name of the API has to be configured via parameter CanIfDispatchUserTrcvModeIndicationName.⌋

9 Sequence diagrams

The following sequence diagrams show the interactions between CanIf and CanDrv.

9.1 Transmit request (single CAN Driver)

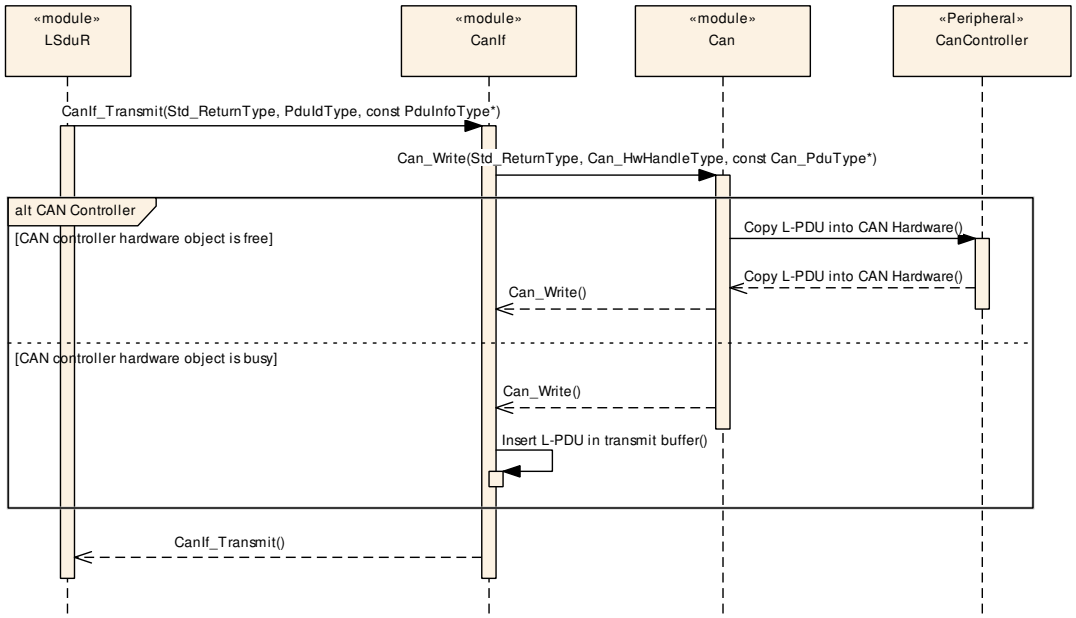


Figure 9.1: Transmission request with a single CAN Driver

Activity	Description
Transmission request	The upper layer initiates a transmit request via the service CanIf_Transmit(). The parameter TxPduId identifies the requested L-SDU. The service performs following steps: • validation of the input parameter • definition of the CAN Controller to be used The second parameter *PduInfoPtr is a pointer on the structure with transmit L-SDU related data such as SduLength and *SduDataPtr.
Start transmission	CanIf_Transmit() requests a transmission and calls the CanDrv service Can_Write() with corresponding processing of the HTH.
Hardware request	Can_Write() writes all L-PDU data in the CAN Hardware (if it is free) and sets the hardware request for transmission.
E_OK from Can_Write service	Can_Write() returns E_OK to CanIf_Transmit().
CAN_BUSY from Can_Write service	If CanDrv detects, there are no free hardware objects available, it returns CAN_BUSY to CanIf.
Copying into the buffer	The L-PDU of the rejected transmit request will be inserted in the transmit buffer of CanIf until the next transmit confirmation.
E_OK from CanIf	CanIf_Transmit() returns E_OK to the upper layer.

9.2 Transmit request (multiple CAN Drivers)

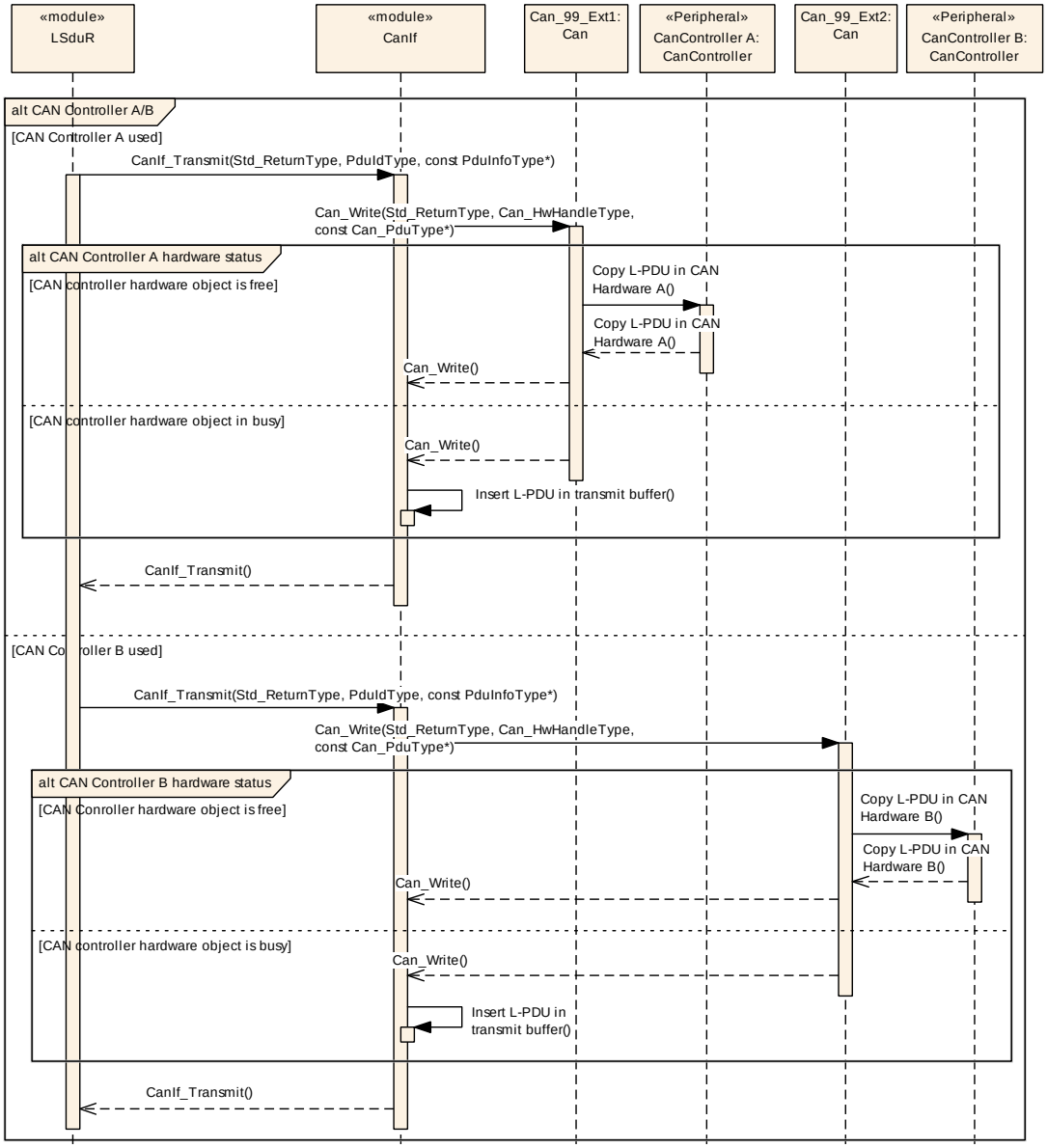


Figure 9.2: Transmission request with multiple CAN Drivers

First transmit request:

Activity	Description
Transmission request A	The upper layer initiates a transmit request via the service CanIf_Transmit(). The parameter TxPduId identifies the requested L-SDU. The service performs following steps: • validation of the input parameter • definition of the CAN Controller to be used (here: Can_99_Ext1) The second parameter *PduInfoPtr is a pointer on the structure with transmit L-SDU related data such as SduLength and *SduDataPtr.
Start transmission	CanIf_Transmit() requests a transmission and calls the CanDrv Can_99_Ext1 service Can_Write_99_Ext1() with corresponding processing of the HTH.
Hardware request	Can_Write_99_Ext1() writes all L-PDU data in the CAN Hardware of Controller A (if it is free) and sets the hardware request for transmission.
E_OK from Can_Write service	Can_Write_99_Ext1() returns E_OK to CanIf_Transmit().
CAN_BUSY from Can_Write service	If CanDrv Can_99_Ext1 detects, there are no free hardware objects available, it returns CAN_BUSY to CanIf.
Copying into the buffer	The L-PDU of the rejected transmit request will be inserted in the transmit buffers of CanIf until the next transmit confirmation.
E_OK from CanIf	CanIf_Transmit() returns E_OK to the upper layer.

Second transmit request:

Activity	Description
Transmission request B	The upper layer initiates a transmit request via the service CanIf_Transmit(). The parameter TxPduId identifies the requested L-SDU. The service performs following steps: • validation of the input parameter • definition of the CAN Controller to be used (here: Can_99_Ext2) The second parameter *PduInfoPtr is a pointer on the structure with transmit L-SDU related data such as SduLength and *SduDataPtr.
Start transmission	CanIf_Transmit() starts a transmission and calls the CanDrv Can_99_Ext2 service Can_Write_99_Ext2() with corresponding processing of the HTH.
Hardware request	Can_Write_99_Ext2() writes all L-PDU data in the CAN Hardware of Controller B (if it is free) and sets the hardware request for transmission.
E_OK from Can_Write service	Can_Write_99_Ext2() returns E_OK to CanIf_Transmit().
CAN_BUSY from Can_Write service	If CanDrv Can_99_Ext2 detects, there are no free hardware objects available, it returns CAN_BUSY to CanIf.
Copying into the buffer	The L-PDU of the rejected transmit request will be inserted in the transmit buffers of CanIf until the next transmit confirmation.
E_OK from CanIf	CanIf_Transmit() returns E_OK to the upper layer.

9.3 Transmit confirmation (interrupt mode)

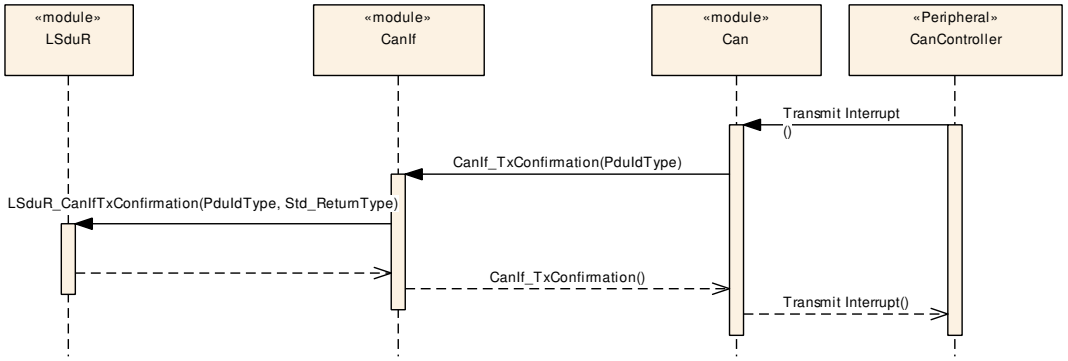


Figure 9.3: Transmit confirmation interrupt driven

Activity	Description
Transmit interrupt	The acknowledged CAN frame signals a successful transmission to the receiving `CAN Controller` and triggers the transmit interrupt.
Confirmation to `CanIf`	`CanDrv` calls the service `CanIf_TxConfirmation()`. The parameter `CanTxPduId` specifies the `L-PDU` previously sent by `Can_Write()`. `CanDrv` must store the all in `HTHs` pending `L-PDU` Ids in an array organized per `HTH` to avoid new search of the `L-PDU` ID for call of `CanIf_TxConfirmation()`.
Confirmation to upper layer	Calling of the upper layer confirmation service `LSduR_CanIfTxConfirmation(id, E_OK)`. It signals a successful `L-SDU` transmission to the upper layer.

9.4 Transmit confirmation (polling mode)

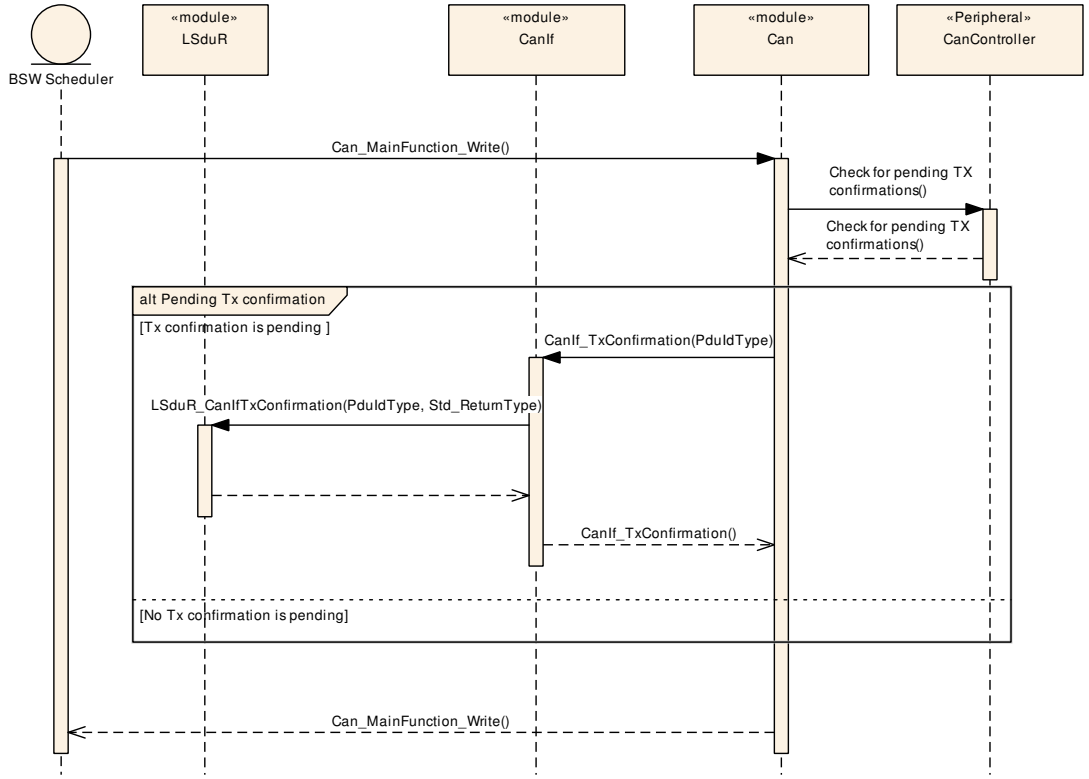


Figure 9.4: Transmit confirmation polling driven

Activity	Description
Cyclic Task CanDrv	The service `Can_MainFunction_Write()` is called by the BSW Scheduler.
Check for pending transmit confirmations	`Can_MainFunction_Write()` checks the underlying `CAN Controller`(s) about pending transmit confirmations of previously succeeded transmit events.
Transmit Confirmation	The acknowledged CAN frame signals a successful transmission to the sending `CAN Controller`.
Confirmation to CanIf	`CanDrv` calls the service `CanIf_TxConfirmation()`. The parameter `CanTxPduId` specifies the `L-PDU` previously sent by `Can_Write()`. `CanDrv` must store the all in `HTHs` pending `L-PDU` Ids in an array organized per `HTH` to avoid new search of the `L-PDU` ID for call of `CanIf_TxConfirmation()`.
Confirmation to upper layer	Calling of the corresponding upper layer confirmation service `LSduR_CanIfTxConfirmation(id, E_OK)`. It signals a successful `L-SDU` transmission to the upper layer.

9.5 Transmit confirmation (with buffering)

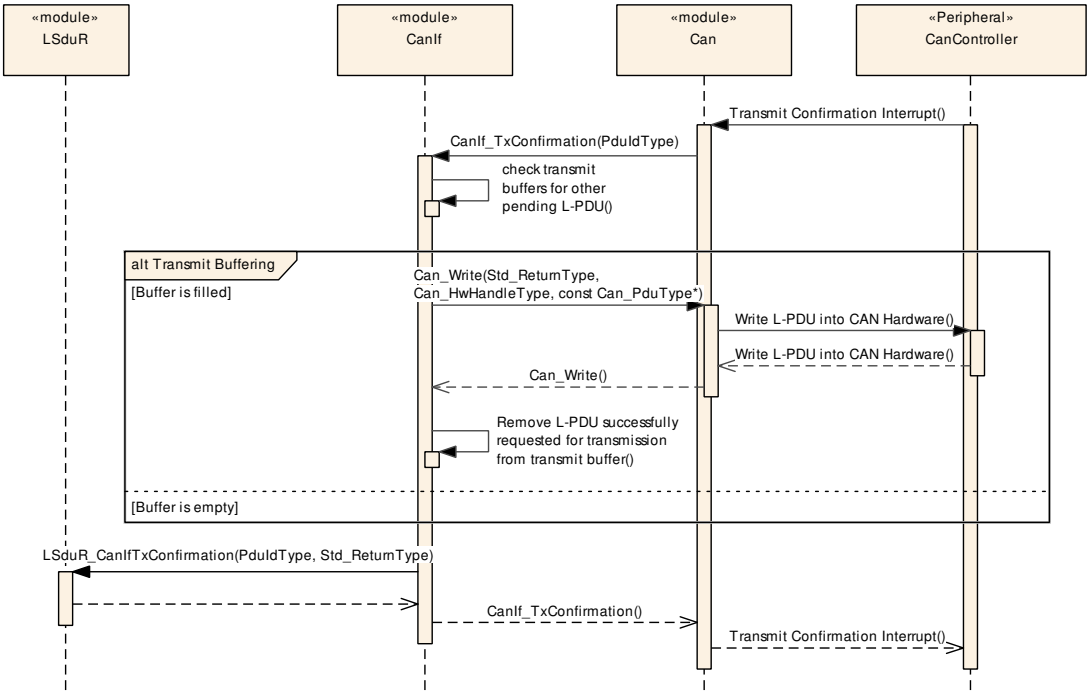


Figure 9.5: Transmit confirmation with buffering

Activity	Description
Transmit interrupt	Acknowledged CAN frame signals successful transmission to receiving CAN Controller and triggers transmit interrupt.
Confirmation to CanIf	CanDrv calls service CanIf_TxConfirmation(). Parameter CanTxPduId specifies the L-PDU previously transmitted by Can_Write(). CanDrv must store the all in HTHs pending L-PDU Ids in an array organized per HTH to avoid new search of the L-PDU ID for call of CanIf_TxConfirmation().
Check of transmit buffers	The transmit buffers of CanIf checked, whether a pending L-PDU is stored or not.
Transmit request passed to CanDrv	In case of pending L-PDUs in the transmit buffers the highest priority order the latest L-PDU is requested for transmission by Can_Write(). It signals a successful L-PDU transmission to the upper layer. Thus Can_Write() can be called re-entrant.
Remove transmitted L-PDU from transmit buffers	The L-PDU pending for transmission is removed from the transmission buffers by CanIf.
Confirmation to the upper layer	Calling of the corresponding upper layer confirmation service LSduR_CanIfTxConfirmation(id, E_OK). It signals a successful L-SDU transmission to the upper layer.

9.6 Trigger Transmit Request

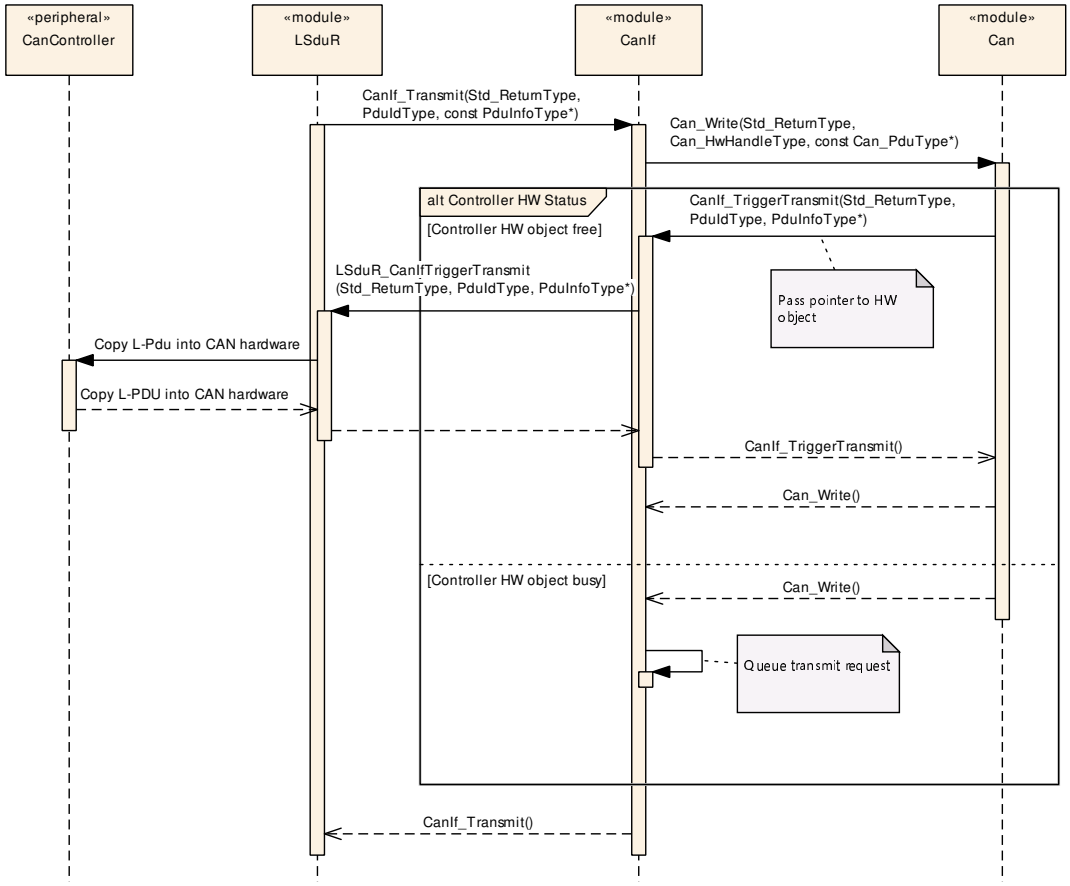


Figure 9.6: Trigger Transmit Request

Activity	Description
Transmission request	The upper layer initiates a transmit request via the service `CanIf_Transmit()`. The parameter `TxPduId` identifies the requested `L-SDU`. The service performs following steps: • validation of the input parameter • definition of the `CAN Controller` to be used The second parameter `*PduInfoPtr` is a pointer to the structure with the size (`SduLength`) of the `L-SDU` to be transmitted. The actual SDU data has not been passed by the upper layer. Hence, the pointer `*SduDataPtr` points to `NULL`.
Start transmission	`CanIf_Transmit()` requests a transmission and calls the `CanDrv` service `Can_Write()` with corresponding processing of the `HTH`.
Trigger transmission	If the CAN hardware is free `Can_Write()` requests the SDU data from `CanIf` by its service `CanIf_TriggerTransmit()` passing the `L-SDUs` corresponding ID and a pointer to the CAN hardware's buffer. `CanIf` forwards the trigger transmit request to the corresponding upper layer (`CanIfUser`). `CanIf` passes the buffer pointer received by `CanDrv`. The `CanIfUser` finally copies the SDU data to the buffer provided by `CanIf` (the CAN hardware buffer) and returns status and number of bytes effectively written.
`E_OK` from `Can_Write()` service	`Can_Write()` returns `E_OK` to `CanIf_Transmit()`.
`CAN_BUSY` from `Can_Write()` service	If `CanDrv` detects, there are no free hardware objects available, it returns `CAN_BUSY` to `CanIf`.
Queuing of transmission request	The `Transmit Request` for the `L-PDU`, which has been rejected by `CanDrv`, is queued by `CanIf` until the next transmit confirmation.
`E_OK` from `CanIf`	`CanIf_Transmit()` returns `E_OK` to the upper layer.

9.7 Receive indication (interrupt mode)

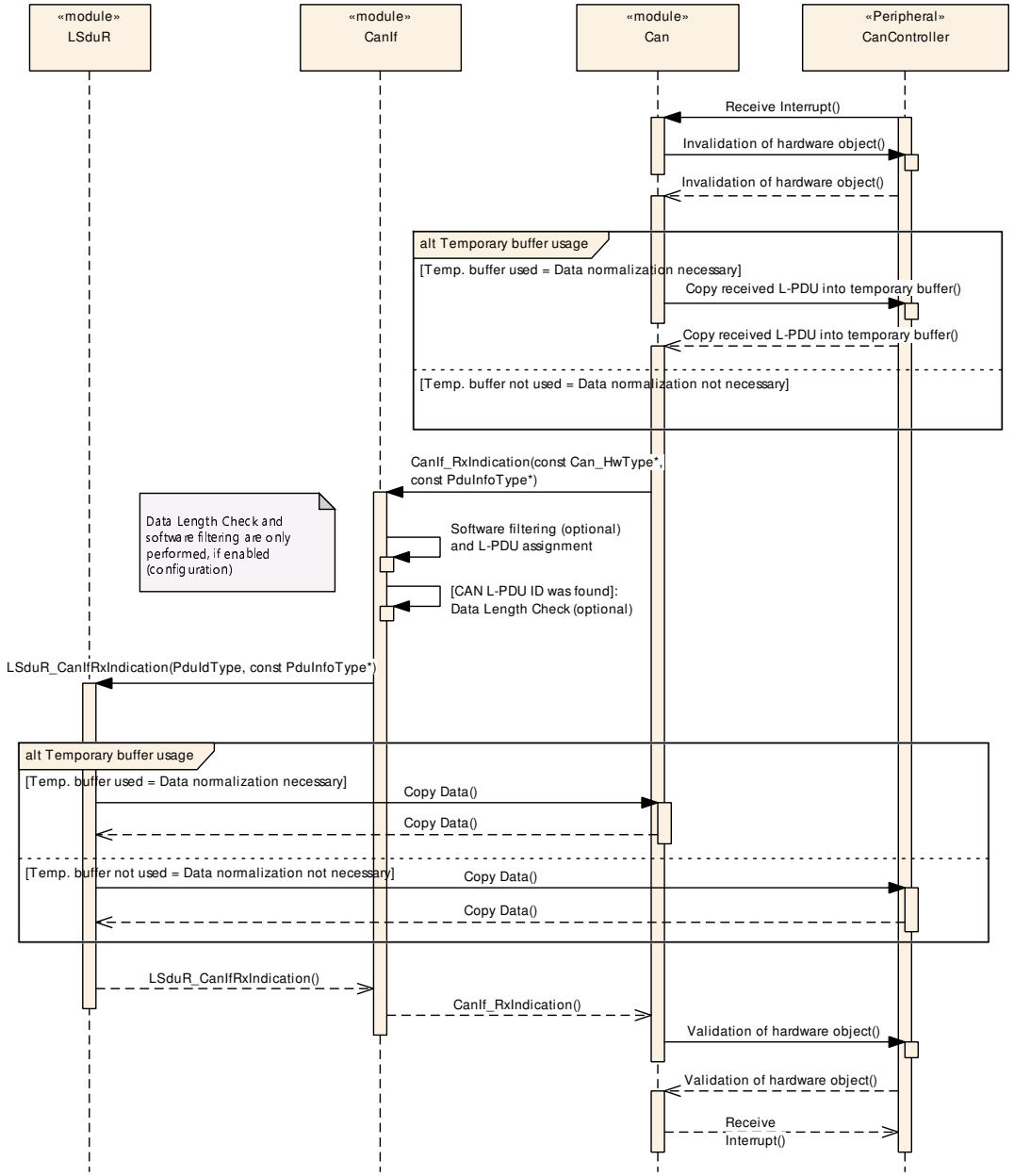


Figure 9.7: Receive indication interrupt driven

Activity	Description
Receive Interrupt	The CAN Controller indicates a successful reception and triggers a receive interrupt.
Invalidation of CAN hardware object, provide CPU access to CAN mailbox	The CPU (CanDrv) get exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received.
Buffering, normalizing	The L-PDU is normalized and is buffered in the temporary buffer located in CanDrv. Each CanDrv owns such a temporary buffer for every Physical Channel only if normalizing of the data is necessary.

Indication to CanIf	The reception is indicated to CanIf by calling of CanIf_RxIndication(). The HRH specifies the CAN RAM Hardware Object and the corresponding CAN Controller, which contains the received L-PDU. The temporary buffer is referenced to CanIf by PduInfoPtr->SduDataPtr.
Software Filtering	The Software Filtering checks, whether the received L-PDU will be processed on a local ECU. If not, the received L-PDU is not indicated to upper layers. Further processing is suppressed.
Data Length Check	If the L-PDU is found, the Data Length of the received L-PDU is compared with the expected, statically configured one for the received L-PDU.
Receive Indication to the upper layer	The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter RxPduId specifies the L-SDU, the second parameter is the reference on the temporary buffer within the L-SDU. During is execution of this service the CAN hardware buffers must be unlocked for CPU access/locked for CAN Controller access.
Validation of CAN hardware object, allow access of CAN Controller to CAN mailbox	The CAN Controller get back exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were already being copied into the upper layer buffer.

9.8 Receive indication (polling mode)

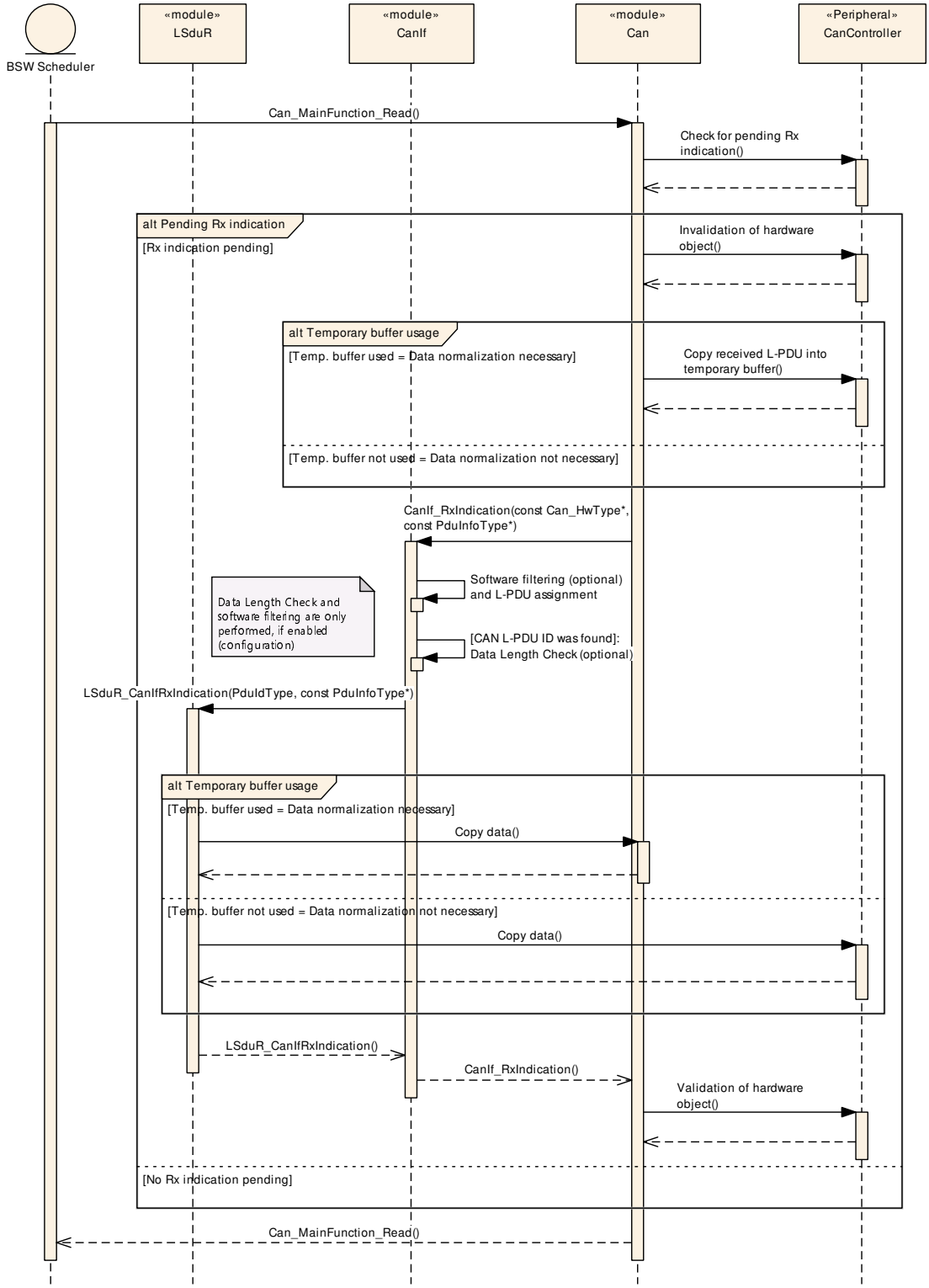


Figure 9.8: Receive indication polling driven

Activity	Description
Cyclic Task CanDrv	The service Can_MainFunction_Read() is called by the BSW Scheduler.
Check for new received L-PDU	Can_MainFunction_Read() checks the underlying CAN Controller(s) about new received L-PDUs.
Invalidation of CAN hardware object, provide CPU access to CAN mailbox	In case of a new receive event the CPU (CanDrv) get exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received.
Buffering, normalizing	In case of a new receive event the L-PDU is normalized and is buffered in the temporary buffer located in CanDrv. Each CanDrv owns such a temporary buffer for every Physical Channel only if normalizing of the data is necessary.
Indication to CanIf	The reception is indicated to CanIf by calling of CanIf_RxIndication(). The HRH specifies the CAN RAM Hardware Object and the corresponding CAN Controller, which contains the received L-PDU. The temporary buffer is referenced to CanIf by PduInfoPtr->SduDataPtr.
Software Filtering	The Software Filtering checks, whether the received L-PDU will be processed on a local ECU. If not, the received L-PDU is not indicated to upper layers. Further processing is suppressed.
Data Length Check	If the L-PDU is found, the Data Length of the received L-PDU is compared with the expected, statically configured one for the received L-PDU.
Receive Indication to the upper layer	If configured, the corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter RxPduId specifies the L-SDU, the second parameter is the reference on the temporary buffer within the L-SDU. During is execution of this service the CAN hardware buffers must be unlocked for CPU access/locked for CAN Controller access.
Validation of CAN hardware object, allow access of CAN Controller to CAN mailbox	The CAN Controller get back exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were already being copied into the upper layer buffer.

9.9 Read received data

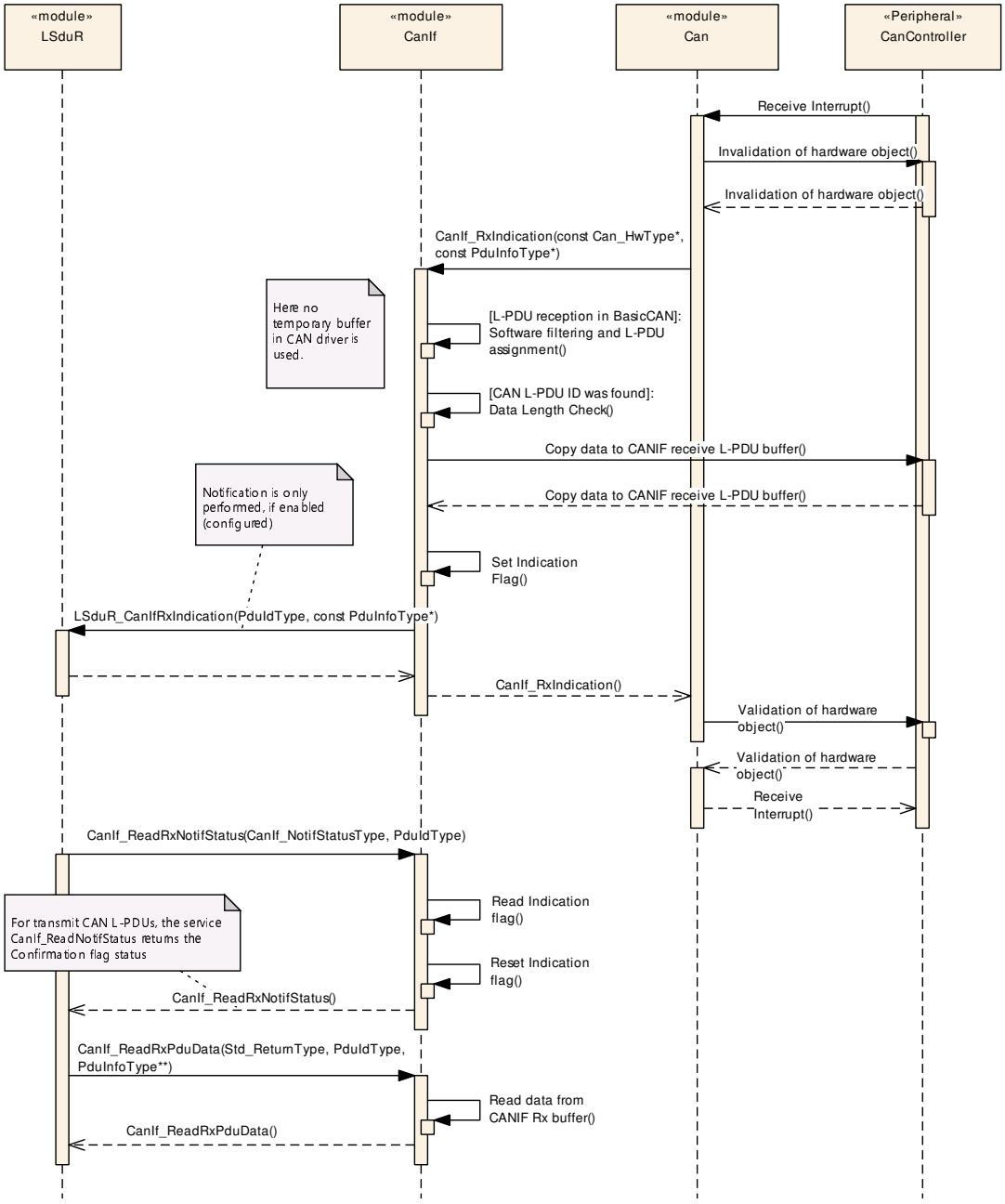


Figure 9.9: Read received data

Activity	Description
Receive Interrupt	The `CAN Controller` indicates a successful reception and triggers a receive interrupt.
Invalidation of CAN hardware object, provide CPU access to CAN mailbox	The CPU (`CanDrv`) get exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received.

Buffering, normalizing	The L-PDU is normalized and is buffered in the temporary buffer located in CanDrv. Each CanDrv owns such a temporary buffer for every Physical Channel only if normalizing of the data is necessary.
Indication to CanIf	The reception is indicated to CanIf by calling of CanIf_RxIndication(). The HRH specifies the CAN RAM Hardware Object and the corresponding CAN Controller, which contains the received L-PDU. The temporary buffer is referenced to CanIf by PduInfoPtr->SduDataPtr.
Software Filtering	The Software Filtering checks, whether the received L-PDU will be processed on a local ECU. If not, the received L-PDU is not indicated to upper layers. Further processing is suppressed.
Data Length Check	If the L-PDU is found, the Data Length of the received L-PDU is compared with the expected, statically configured one for the received L-PDU.
Copy data	The data is copied out of the CAN hardware into the receive CAN L-PDU buffers in CanIf. During access the CAN hardware buffers must be unlocked for CPU access/locked for CAN Controller access.
Indication Flag	Set indication status flag for the received L-PDU in CanIf.
Receive Indication to the upper layer	The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter RxPduId specifies the L-SDU, the second parameter is the reference on the temporary buffer within the L-SDU.
Validation of CAN hardware object, allow access of CAN Controller to CAN mailbox	The CAN Controller get back exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were already being copied into the upper layer buffer.
Read indication status	Times later the upper layer can read the indication status by call of CanIf_ReadRxNotifStatus(). This service can also be used for transmit L-PDUs. Then it return the confirmation status.
Reset indication status	Before CanIf_ReadRxNotifStatus() returns, the indication status is reset.
Read received data	Times later the upper layer can read the received data by call of CanIf_ReadRxPduData().
Read CanIf Rx buffer	CanIf_ReadRxPduData() reads the data from CanIf Rx buffer.
E_OK from CanIf	If CanIf_ReadRxPduData() was successful, the request returns E_OK with valid CanIfRxInfoPtr.

9.10 Start CAN network

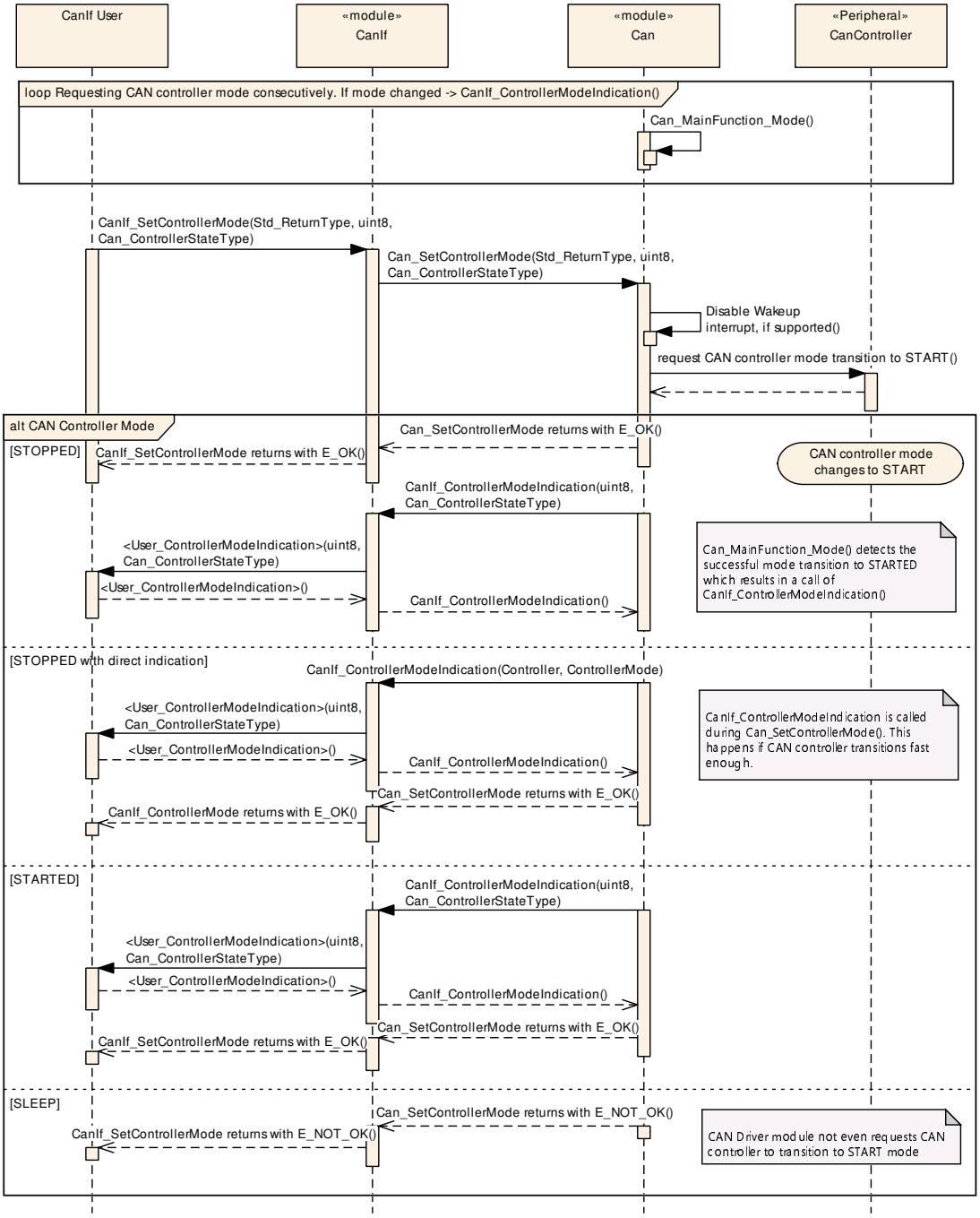


Figure 9.10: Start CAN network

This sequence diagram resembles "Stop CAN network" or "Sleep CAN network".

Activity	Description
Loop requesting CAN controller mode consecutively.	The `Can_MainFunction_Mode()` is triggered consecutively. It checks the HW if a controller mode has changed. If so, it is notified via a function call of `CanIf_ControllerModeIndication(Controller, ControllerMode)`.
The upper layer requests "`STARTED`" mode of the desired CAN controller	The upper layer calls `CanIf_SetControllerMode(ControllerId, CAN_CS_STARTED)` to request `STARTED` mode for the requested CAN controller.
CanDrv disables wake up interrupts, if supported	This is only done in case of requesting "`STARTED`" mode. If "`SLEEP`" mode of CAN controller is requested, here the wake up interrupts are enabled. In case of "`STOPPED`", nothing happens.
CanDrv requests the CAN controller to transition into the requested mode (`CAN_CS_STARTED`).	During function call `Can_SetControllerMode(Controller, Can_ControllerStateType)`, the CanDrv enters the request into the hardware of the CAN controller. This may mean that the controller mode transitions directly, but it could mean that it takes a few milliseconds until the controller changes its state. It depends on the controllers.
The following reaction depends on the controller and its current operation mode	
CAN controller was in `STOPPED` mode	The former request `Can_SetControllerMode()` returns and informs CanIf about a successful request which in turn returns the upper layer request `CanIf_SetControllerMode()`. The `Can_MainFunction_Mode()` detects the successful mode transition of the CAN controller and inform the CanIf asynchronously via `CanIf_ControllerModeIndication(Controller, CAN_CS_STARTED)`.
CAN controller was in `STOPPED` mode and the CAN controller transitions very fast so that mode indication is called during transition request	During the former request `Can_SetControllerMode()` the function `CanIf_ControllerModeIndication(Controller, CAN_CS_STARTED)` is called to inform the CanIf directly about the successful mode transition. When `CanIf_ControllerModeIndication(Controller, CAN_CS_STARTED)` returned, the request `Can_SetControllerMode()` returns and informs CanIf about a successful request which in turn returns the upper layer request `CanIf_SetControllerMode()`.
CAN controller was in `STARTED` mode	During the former request `Can_SetControllerMode()` the function `CanIf_ControllerModeIndication(Controller, CAN_CS_STARTED)` is called to inform the CanIf directly about the successful mode transition (because the mode was already started). When `CanIf_ControllerModeIndication(Controller, CAN_CS_STARTED)` returned, the request `Can_SetControllerMode()` returns and informs CanIf about a successful request which in turn returns the upper layer request `CanIf_SetControllerMode()`.
CAN controller was in `SLEEP` mode	This transition is not allowed -> `E_NOT_OK`.

9.11 BusOff notification

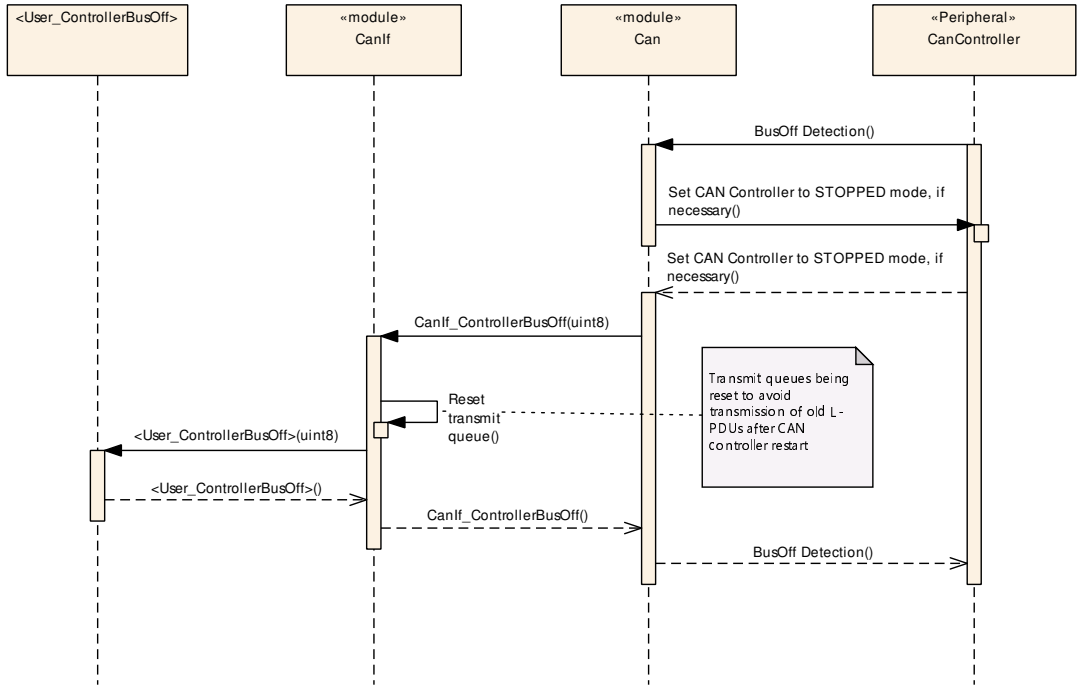


Figure 9.11: BusOff notification

Activity	Description
BusOff detection interrupt	The CAN controller signals a BusOff event.
Stop CAN controller	CAN controller is set to `STOPPED` mode by the CAN Driver, if necessary.
BusOff indication to CAN Interface	BusOff is notified to the CanIf by calling of `CanIf_ControllerBusOff()`
BusOff indication to upper layer (CanSM)	BusOff is notified to the upper layer by calling of `<User_ControllerBusOff>()`

9.12 BusOff recovery

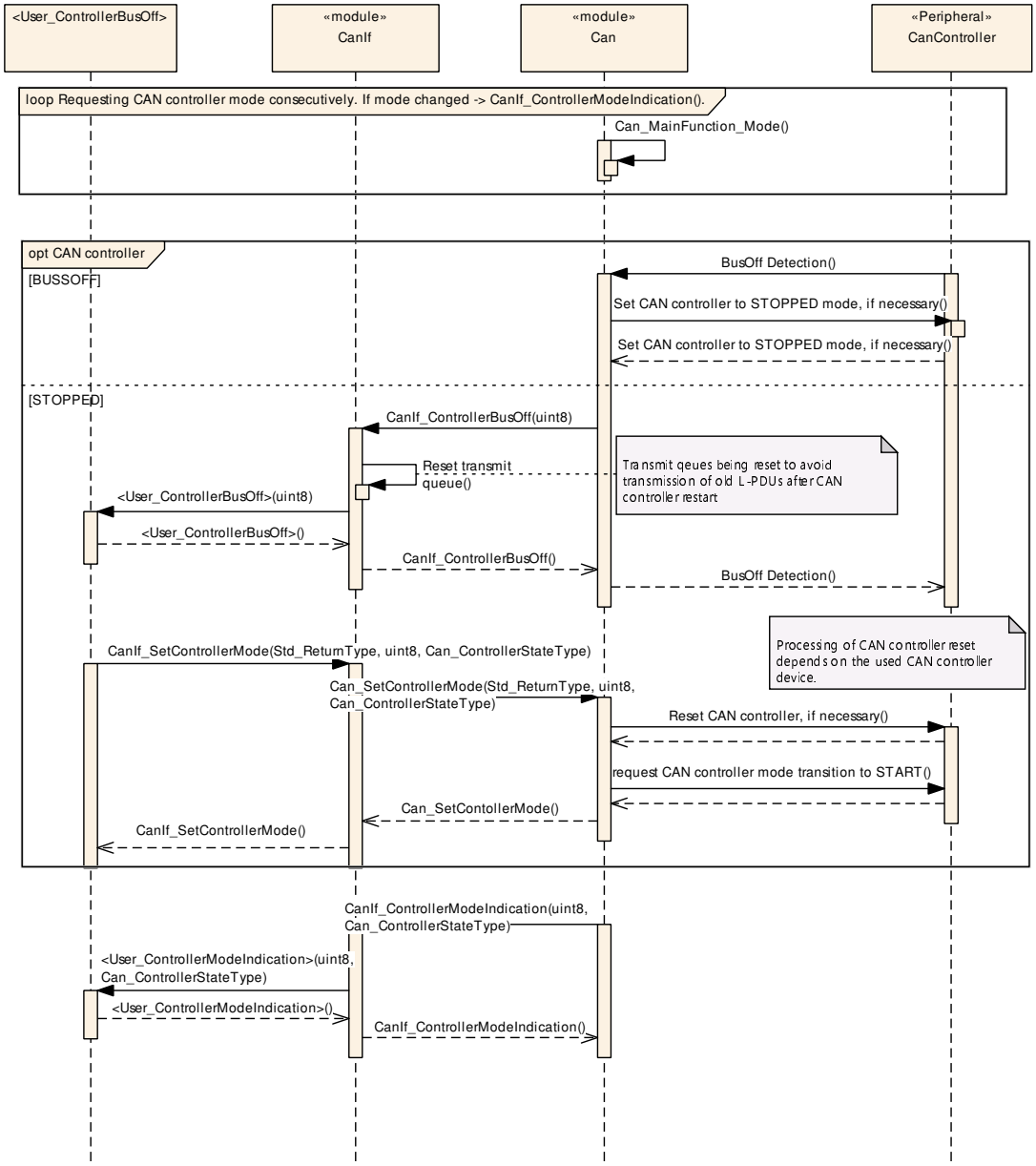


Figure 9.12: BusOff recovery

Activity	Description
BusOff detection interrupt	The CAN controller signals a BusOff event.
Stop CAN controller	CAN controller is set to STOPPED mode by the CanDrv, if necessary
BusOff indication to CanIf	BusOff is notified to the CanIf by calling of CanIf_ControllerBusOff(). The transmit buffers inside CanIf will be reset.
BusOff indication to upper layer	BusOff is notified to the upper layer by calling of <User_ControllerBusOff>()
Upper Layer (CanSM) initiates BusOff Recovery	After a time specified by the BusOff Recovery algorithm the Recovery process itself in initiated by CanIf_SetControllerMode(ControllerId, CAN_CS_STARTED).
Restart of CAN controller	The driver restarts the CAN controller by call of Can_SetControllerMode(Controller, CAN_CS_STARTED).
CAN controller started	CanDrv informs CanIf about the successful start by calling CanIf_ControllerModeIndication(). CanIf informs in turn upper layers about the mode change.

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. For general information about the definition of containers and parameters, refer to [8] Chapter 10.1 *"Introduction to configuration specification"*.

Chapter 10.1 specifies the structure (containers) and the parameters of the CanIf.

10.1 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 "Functional specification" and Chapter 8 "API specification".

The listed configuration items can be derived from a network description database, which is based on the EcuConfigurationTemplate. The configuration tool shall extract all information to configure the `CanIf`.

The CanIf has access to the `CanDrv` configuration data. All public CanDrv configuration data are described in [1, Specification of CAN Driver].

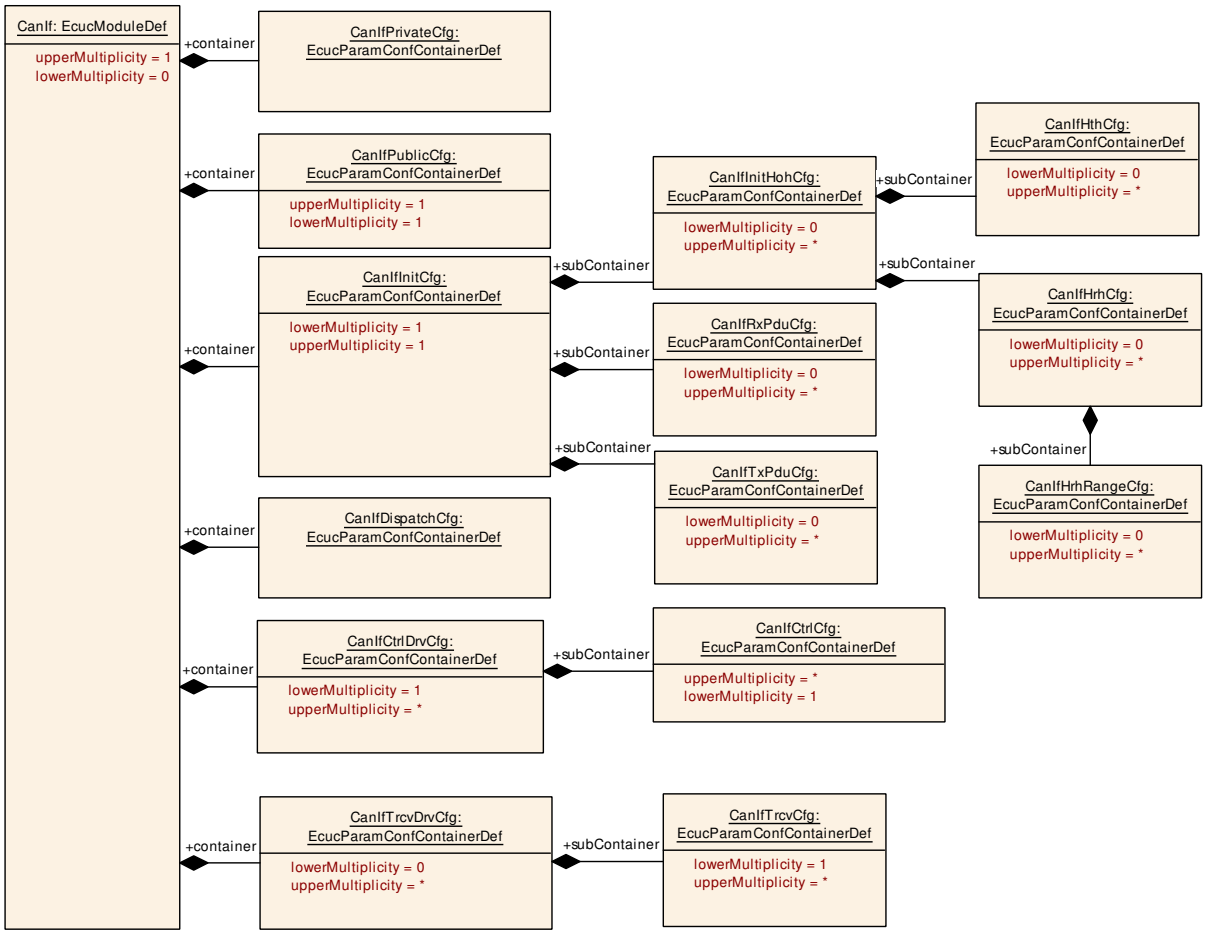


Figure 10.1: Overview about CAN Interface configuration containers

10.1.1 CanIf

[ECUC_CanIf_00244] Definition of EcucModuleDef CanIf ⌈

Module Name	CanIf
Description	This container includes all necessary configuration sub-containers according the CAN Interface configuration structure.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfCtrlDrvCfg	1..*	Configuration parameters for all the underlying CAN Driver modules are aggregated under this container. For each CAN Driver module a seperate instance of this container has to be provided.
CanIfDispatchCfg	1	Callback functions provided by upper layer modules of the CanIf. The callback functions defined in this container are common to all configured CAN Driver / CAN Transceiver Driver modules.
CanIfInitCfg	1	This container contains the init parameters of the CAN Interface.
CanIfPrivateCfg	1	This container contains the private configuration (parameters) of the CAN Interface.
CanIfPublicCfg	1	This container contains the public configuration (parameters) of the CAN Interface.
CanIfTrcvDrvCfg	0..*	This container contains the configuration (parameters) of all addressed CAN transceivers by each underlying CAN Transceiver Driver module. For each CAN transceiver Driver a seperate instance of this container shall be provided.

⌋

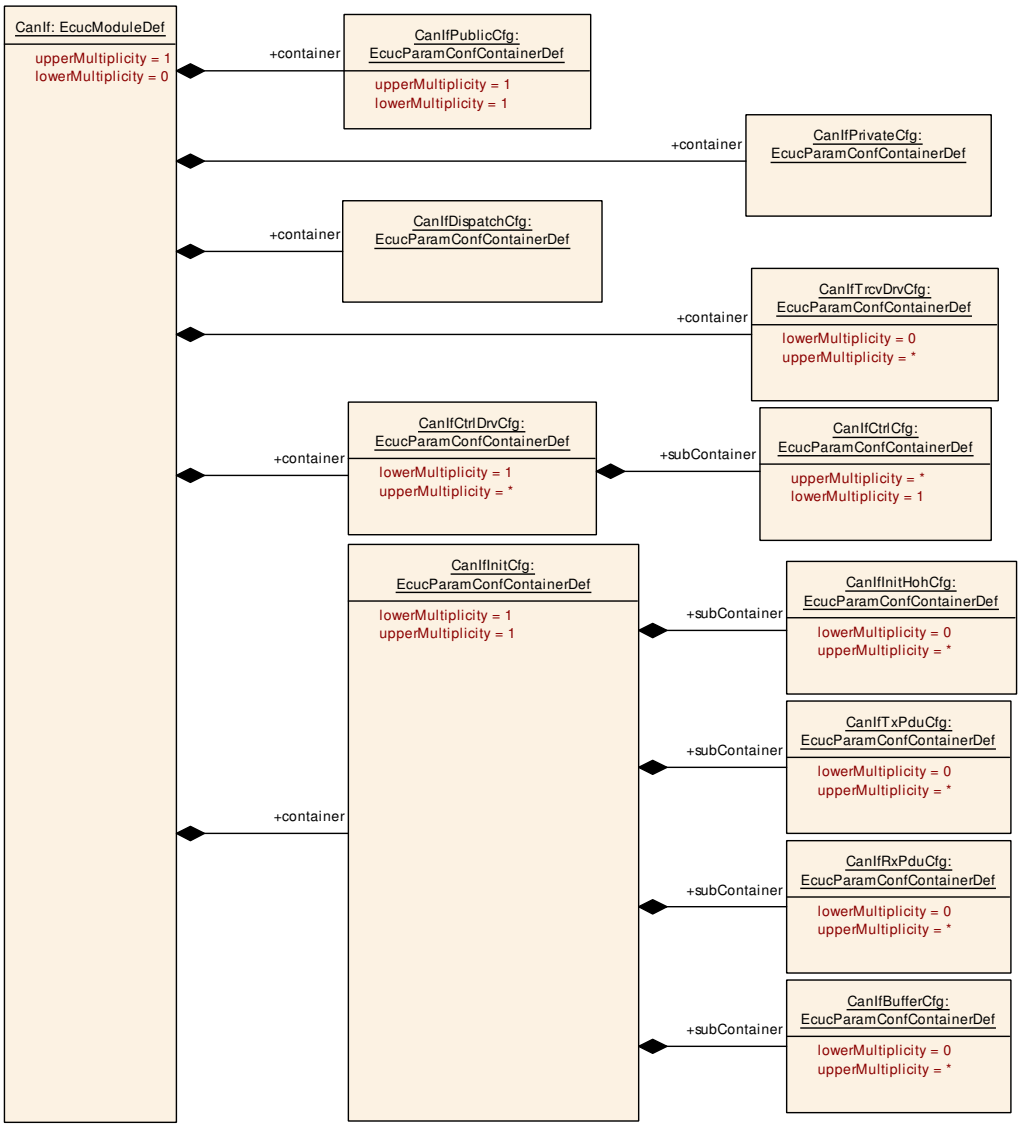


Figure 10.2: AR_EcucDef_CanIf

10.1.2 CanIfPrivateCfg

[ECUC_CanIf_00245] Definition of EcucParamConfContainerDef CanIfPrivateCfg

⌈

Container Name	CanIfPrivateCfg
Parent Container	CanIf
Upstream mapping defined	–
Description	This container contains the private configuration (parameters) of the CAN Interface.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfFixedBuffer	1	[ECUC_CanIf_00827]
CanIfPrivateDataLengthCheck	1	[ECUC_CanIf_00617]
CanIfPrivateSoftwareFilterType	1	[ECUC_CanIf_00619]

No Included Containers

⌋

[ECUC_CanIf_00827] Definition of EcucBooleanParamDef CanIfFixedBuffer ⌈

Parameter Name	CanIfFixedBuffer
Parent Container	CanIfPrivateCfg
Upstream mapping defined	–
Description	This parameter defines if the buffer element length shall be fixed to 8 Bytes for buffers to which only PDUs < 8 Bytes are assigned. TRUE: Minimum buffer element length is fixed to 8 Bytes. FALSE: Buffer element length depends on the configured length of the referenced global PDUs (see ECUC_EcuC_00078).
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00617] Definition of EcucBooleanParamDef CanIfPrivateDataLengthCheck ⌈

Parameter Name	CanIfPrivateDataLengthCheck
Parent Container	CanIfPrivateCfg
Upstream mapping defined	–
Description	Selects whether Data Length Check is supported. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true
Pre-Compile only	true

⌋

[ECUC_CanIf_00619] Definition of EcucEnumerationParamDef CanIfPrivateSoftwareFilterType ⌈

Parameter Name	CanIfPrivateSoftwareFilterType
Parent Container	CanIfPrivateCfg
Upstream mapping defined	–

▽

△

Description	Selects the desired software filter mechanism for reception only. Each implemented software filtering method is identified by this enumeration number. Range: Types implemented software filtering methods Dependency: BasicCAN reception must be enabled by referenced parameter Can HandleType of the CAN Driver module via CanIfHrhIdSymRef for at least one HRH.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	BINARY	Selects Binary Filter method.
	INDEX	Selects Index Filter method.
	LINEAR	Selects Linear Filter method.
	TABLE	Selects Table Filter method.
Pre-Compile only	true	

⌋

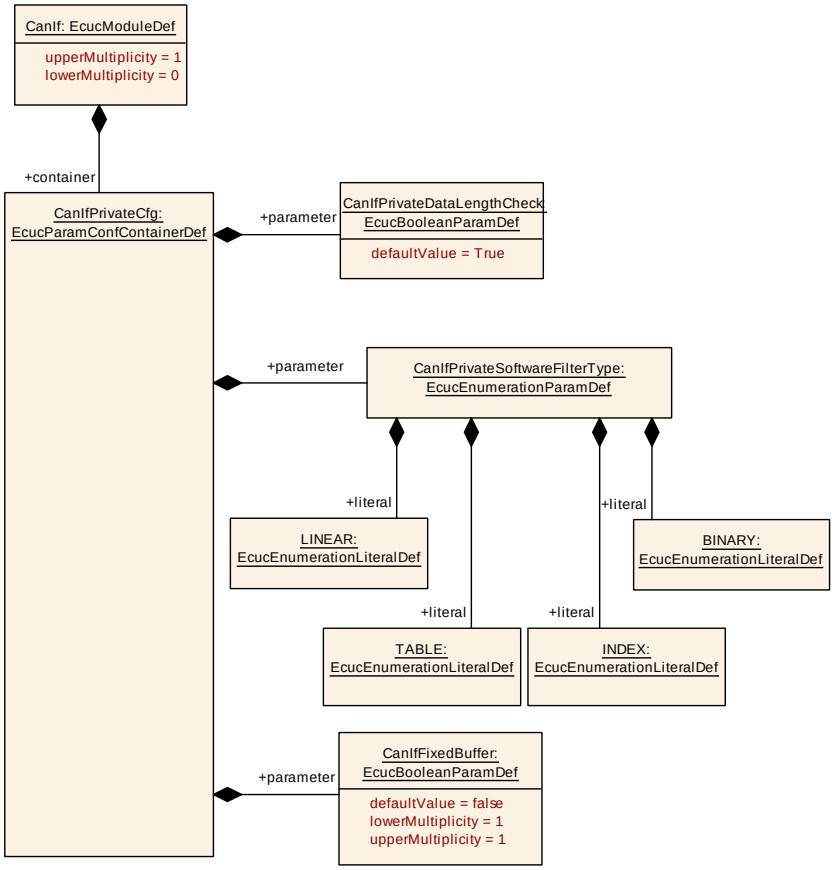


Figure 10.3: AR_EcucDef_CanIfPrivateCfg

10.1.3 CanIfPublicCfg

[ECUC_CanIf_00246] Definition of EcucParamConfContainerDef CanIfPublicCfg

⌈

Container Name	CanIfPublicCfg
Parent Container	CanIf
Upstream mapping defined	–
Description	This container contains the public configuration (parameters) of the CAN Interface.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfBusMirroringSupport	1	[ECUC_CanIf_00847]
CanIfDevErrorDetect	1	[ECUC_CanIf_00614]
CanIfEnableSecurityEventReporting	1	[ECUC_CanIf_00848]
CanIfGlobalTimeSupport	1	[ECUC_CanIf_00854]
CanIfMetaDataSupport	0..1	[ECUC_CanIf_00824]
CanIfPublicCddHeaderFile	0..*	[ECUC_CanIf_00671]
CanIfPublicCtrlPnEnable	1	[ECUC_CanIf_00866]
CanIfPublicHandleTypeEnum	1	[ECUC_CanIf_00742]
CanIfPublicMultipleDrvSupport	1	[ECUC_CanIf_00612]
CanIfPublicPnSupport	1	[ECUC_CanIf_00772]
CanIfPublicReadRxPduDataApi	1	[ECUC_CanIf_00607]
CanIfPublicReadRxPduNotifyStatusApi	1	[ECUC_CanIf_00608]
CanIfPublicReadTxPduNotifyStatusApi	1	[ECUC_CanIf_00609]
CanIfPublicSetDynamicTxIdApi	1	[ECUC_CanIf_00610]
CanIfPublicTrcvPnEnable	1	[ECUC_CanIf_00865]
CanIfPublicTxBuffering	1	[ECUC_CanIf_00618]
CanIfPublicTxConfirmPollingSupport	1	[ECUC_CanIf_00733]
CanIfPublicWakeupCheckValidByNM	0..1	[ECUC_CanIf_00741]
CanIfPublicWakeupCheckValidSupport	1	[ECUC_CanIf_00611]
CanIfSetBaudrateApi	0..1	[ECUC_CanIf_00838]
CanIfTriggerTransmitSupport	1	[ECUC_CanIf_00844]
CanIfTxOfflineActiveSupport	1	[ECUC_CanIf_00837]
CanIfWakeupSupport	1	[ECUC_CanIf_00843]

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfSecurityEventRefs	0..1	Container for the references to IdsMEvent elements representing the security events that the CanIf module shall report to the IdsM in case the coresponding security related event occurs (and if CanIfEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events. **Tags:** atp.Status=draft

⌋

[ECUC_CanIf_00847] Definition of EcucBooleanParamDef CanIfBusMirroring Support ⌈

Parameter Name	CanIfBusMirroringSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enable support for Bus Mirroring.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00614] Definition of EcucBooleanParamDef CanIfDevErrorDetect ⌈

Parameter Name	CanIfDevErrorDetect
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00848] Definition of EcucBooleanParamDef CanIfEnableSecurityEventReporting

Status: DRAFT

⌈

Parameter Name	CanIfEnableSecurityEventReporting
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Switches the reporting of security events to the IdsM: - true: reporting is enabled. - false: reporting is disabled. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00854] Definition of EcucBooleanParamDef CanIfGlobalTimeSupport

Status: DRAFT

⌈

Parameter Name	CanIfGlobalTimeSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables/Disables the Global Time APIs used when hardware timestamping is supported. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00824] Definition of EcucBooleanParamDef CanIfMetaDataSupport ⌈

Parameter Name	CanIfMetaDataSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enable support for dynamic ID handling using L-SDU MetaData.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00671] Definition of EcucStringParamDef CanIfPublicCddHeaderFile ⌈

Parameter Name	CanIfPublicCddHeaderFile
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Defines header files for callback functions which shall be included in case of CDDs. Range of characters is 1.. 32.
Multiplicity	0..*
Type	EcucStringParamDef
Default value	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00866] Definition of EcucBooleanParamDef CanIfPublicCtrlPnEnable ⌈

Parameter Name	CanIfPublicCtrlPnEnable
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Indicates whether Partial Network function is enabled or disabled in CanDrv. • True: Enabled • False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00742] Definition of EcucEnumerationParamDef CanIfPublicHandleTypeEnum ⌈

Parameter Name	CanIfPublicHandleTypeEnum	
Parent Container	CanIfPublicCfg	
Upstream mapping defined	–	
Description	This parameter is used to configure the Can_HwHandleType. The Can_HwHandleType represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects the extended range shall be used (UINT16). Dependency: Can_HwHandleType	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	UINT16	Type of the data is uint16.
	UINT8	Type of the data is uint8.
Pre-Compile only	true	

⌋

[ECUC_CanIf_00612] Definition of EcucBooleanParamDef CanIfPublicMultipleDrvSupport ⌈

Parameter Name	CanIfPublicMultipleDrvSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Selects support for multiple CAN Drivers. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true
Pre-Compile only	true

⌋

[ECUC_CanIf_00772] Definition of EcucBooleanParamDef CanIfPublicPnSupport ⌈

Parameter Name	CanIfPublicPnSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Selects support of Partial Network features in CanIf. • True: Enabled • False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00607] Definition of EcucBooleanParamDef CanIfPublicReadRxPduDataApi ⌈

Parameter Name	CanIfPublicReadRxPduDataApi
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables / Disables the API CanIf_ReadRxPduData() for reading received L-SDU data. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00608] Definition of EcucBooleanParamDef CanIfPublicReadRxPduNotifyStatusApi ⌈

Parameter Name	CanIfPublicReadRxPduNotifyStatusApi
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables and disables the API for reading the notification status of receive L-PDUs. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00609] Definition of EcucBooleanParamDef CanIfPublicReadTxPduNotifyStatusApi ⌈

Parameter Name	CanIfPublicReadTxPduNotifyStatusApi
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables and disables the API for reading the notification status of transmit L-PDUs. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00610] Definition of EcucBooleanParamDef CanIfPublicSetDynamicTxIdApi ⌈

Parameter Name	CanIfPublicSetDynamicTxIdApi
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables and disables the API for reconfiguration of the CAN Identifier for each Transmit L-PDU. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00865] Definition of EcucBooleanParamDef CanIfPublicTrcvPnEnable ⌈

Parameter Name	CanIfPublicTrcvPnEnable
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Indicates whether Partial Network function is enabled or disabled in CanTrcv. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00618] Definition of EcucBooleanParamDef CanIfPublicTxBuffering ⌈

Parameter Name	CanIfPublicTxBuffering
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables and disables the buffering of transmit L-PDUs (rejected by the CanDrv) within the CAN Interface module. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00733] Definition of EcucBooleanParamDef CanIfPublicTxConfirmPollingSupport ⌈

Parameter Name	CanIfPublicTxConfirmPollingSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Configuration parameter to enable/disable the API to poll for Tx Confirmation state. Dependency: CAN State Manager module
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00741] Definition of EcucBooleanParamDef CanIfPublicWakeupCheckValidByNM ⌈

Parameter Name	CanIfPublicWakeupCheckValidByNM
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	If enabled, only NM messages shall validate a detected wake-up event in CanIf. If disabled, all received messages corresponding to a configured Rx PDU shall validate such a wake-up event. This parameter depends on CanIfPublicWakeupCheckValid Support and shall only be configurable, if it is enabled. True: Enabled False: Disabled Dependency: CanIfPublicWakeupCheckValidSupport
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00611] Definition of EcucBooleanParamDef CanIfPublicWakeup CheckValidSupport ⌈

Parameter Name	CanIfPublicWakeupCheckValidSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Selects support for wake up validation True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00838] Definition of EcucBooleanParamDef CanIfSetBaudrateApi ⌈

Parameter Name	CanIfSetBaudrateApi
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Configuration parameter to enable/disable the CanIf_SetBaudrate API to change the baud rate of a CAN Controller. If this parameter is set to true the CanIf_SetBaudrate API shall be supported. Otherwise the API is not supported.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00844] Definition of EcucBooleanParamDef CanIfTriggerTransmit Support ⌈

Parameter Name	CanIfTriggerTransmitSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables the CanIf_TriggerTransmit API at Pre-Compile-Time. Therefore, this parameter defines if there shall be support for trigger transmit transmissions. TRUE: Enabled FALSE: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true
Pre-Compile only	true

⌋

[ECUC_CanIf_00837] Definition of EcucBooleanParamDef CanIfTxOfflineActive Support ⌈

Parameter Name	CanIfTxOfflineActiveSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Determines wether TxOffLineActive feature (see SWS_CANIF_00072) is supported by CanIf. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00843] Definition of EcucBooleanParamDef CanIfWakeupSupport ⌈

Parameter Name	CanIfWakeupSupport
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Enables the CanIf_CheckWakeup API at Pre-Compile-Time. Therefore, this parameter defines if there shall be support for wake-up. TRUE: Enabled FALSE: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true
Pre-Compile only	true

⌋

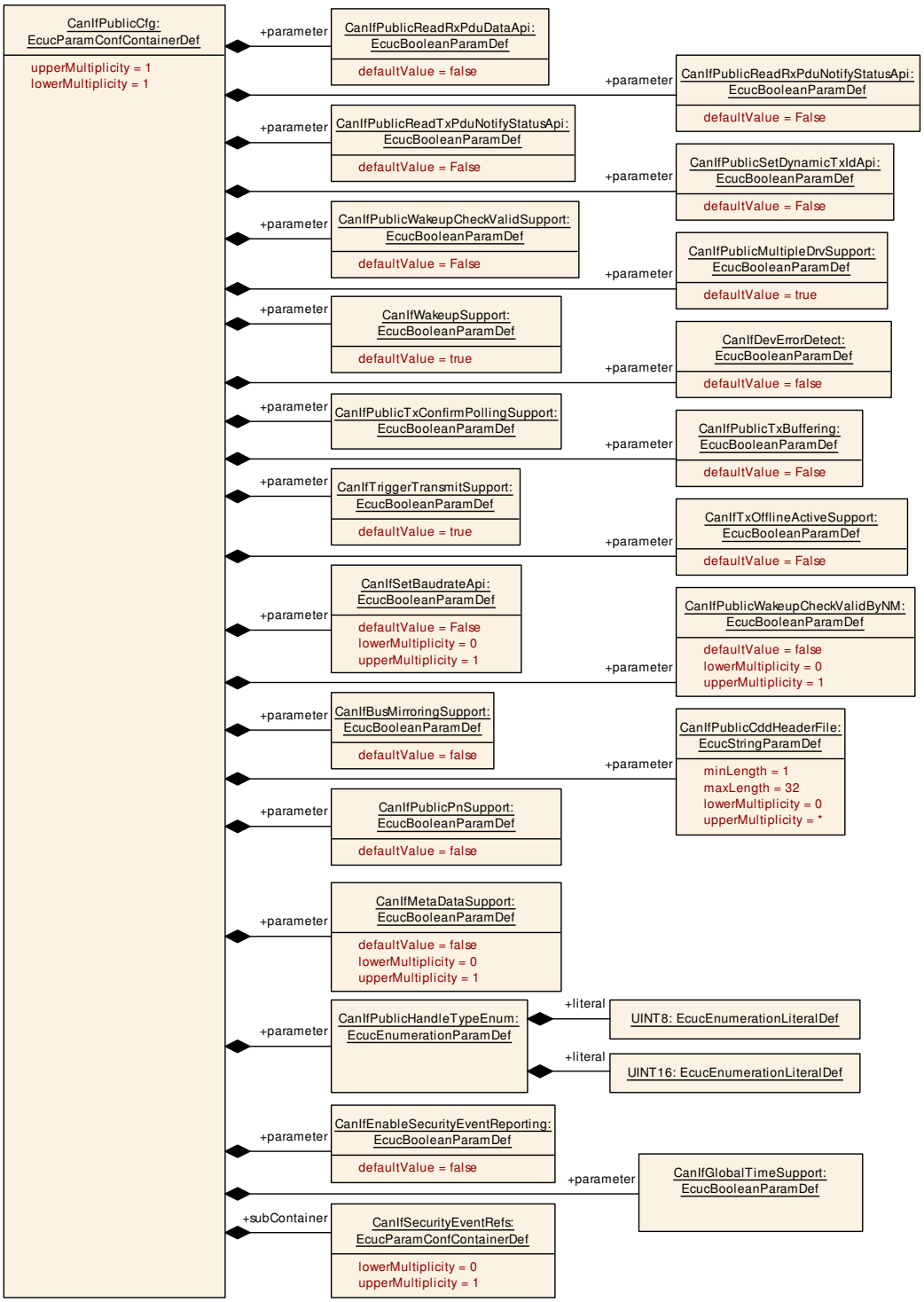


Figure 10.4: AR_EcucDef_CanIfPublicCfg

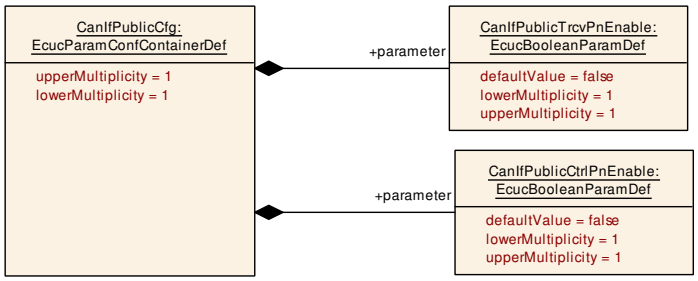


Figure 10.5: AR_EcucDef_CanIfPublicCfg2

10.1.4 CanIfInitCfg

[ECUC_CanIf_00247] Definition of EcucParamConfContainerDef CanIfInitCfg ⌈

Container Name	CanIfInitCfg
Parent Container	CanIf
Upstream mapping defined	–
Description	This container contains the init parameters of the CAN Interface.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfInitCfgSet	1	[ECUC_CanIf_00623]
CanIfMaxBufferSize	0..1	[ECUC_CanIf_00828]
CanIfMaxRxPduCfg	0..1	[ECUC_CanIf_00830]
CanIfMaxTxPduCfg	0..1	[ECUC_CanIf_00829]

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfBufferCfg	0..*	This container contains the Txbuffer configuration. Multiple buffers with different sizes could be configured. If CanIfBuffer Size (ECUC_CanIf_00834) equals 0, the CanIf Tx L-PDU only refers via this CanIfBufferCfg the corresponding CanIfHthCfg.
CanIfInitHohCfg	0..*	This container contains the references to the configuration setup of each underlying CAN Driver.
CanIfRxPduCfg	0..*	This container contains the configuration (parameters) of each receive CAN L-PDU. The SHORT-NAME of "CanIfRxPduConfig" container itself represents the symolic name of Receive L-PDU. This L-SDU produces a meta data item of type CAN_ID_32 and TIMETUPLE_TYPE_PTR, and, depending on CanIfRxPdu XLParams, meta data items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32.

▽

△

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfTxPduCfg	0..*	This container contains the configuration (parameters) of a transmit CAN L-PDU. It has to be configured as often as a transmit CAN L-PDU is needed. The SHORT-NAME of "CanIfTxPduConfig" container represents the symolic name of Transmit L-PDU. This L-SDU consumes a meta data item of type CAN_ID_32, and, depending on CanIfTxPduXLParams, meta data items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32.

⌋

[ECUC_CanIf_00623] Definition of EcucStringParamDef CanIfInitCfgSet ⌈

Parameter Name	CanIfInitCfgSet	
Parent Container	CanIfInitCfg	
Upstream mapping defined	–	
Description	Selects the CAN Interface specific configuration setup. This type of the external data structure shall contain the post build initialization data for the CAN Interface for all underlying CAN Dirvers. constant to CanIf_ConfigType	
Multiplicity	1	
Type	EcucStringParamDef	
Default value	–	
Length	1-32	
Regular Expression	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00828] Definition of EcucIntegerParamDef CanIfMaxBufferSize ⌈

Parameter Name	CanIfMaxBufferSize	
Parent Container	CanIfInitCfg	
Upstream mapping defined	–	
Description	Maximum total size of all Tx buffers. This parameter is needed only in case of post-build loadable implementation using static memory allocation.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_CanIf_00830] Definition of EcucIntegerParamDef CanIfMaxRxPduCfg ⌈

Parameter Name	CanIfMaxRxPduCfg	
Parent Container	CanIfInitCfg	
Upstream mapping defined	–	
Description	Maximum number of Pdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_CanIf_00829] Definition of EcucIntegerParamDef CanIfMaxTxPduCfg ⌈

Parameter Name	CanIfMaxTxPduCfg	
Parent Container	CanIfInitCfg	
Upstream mapping defined	–	
Description	Maximum number of Pdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	
Default value	–	
Pre-Compile only	true	

⌋

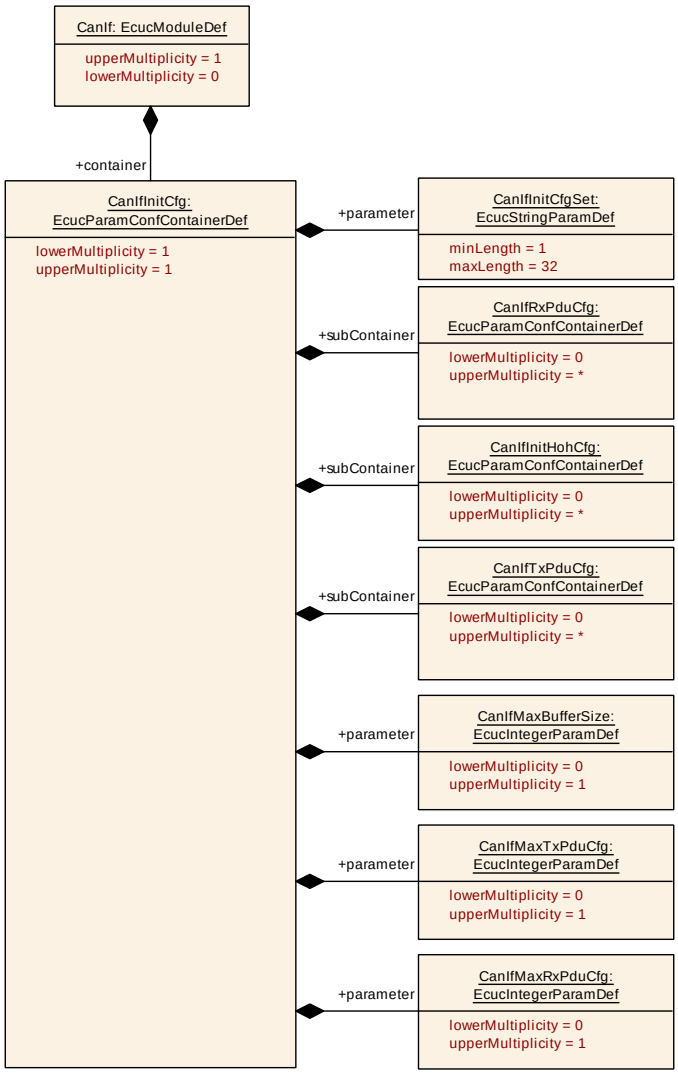


Figure 10.6: AR_EcucDef_CanIfInitCfg

10.1.5 CanIfTxPduCfg

[ECUC_CanIf_00248] Definition of EcucParamConfContainerDef CanIfTxPduCfg ⌈

Container Name	CanIfTxPduCfg
Parent Container	CanIfInitCfg
Upstream mapping defined	–

▽

△

Description	This container contains the configuration (parameters) of a transmit CAN L-PDU. It has to be configured as often as a transmit CAN L-PDU is needed. The SHORT-NAME of "CanIfTxPduConfig" container represents the symolic name of Transmit L-PDU. This L-SDU consumes a meta data item of type CAN_ID_32, and, depending on CanIf TxPduXLParams, meta data items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfTxPduCanId	0..1	[ECUC_CanIf_00592]
CanIfTxPduCanIdMask	0..1	[ECUC_CanIf_00823]
CanIfTxPduId	1	[ECUC_CanIf_00591]
CanIfTxPduPnFilterPdu	0..1	[ECUC_CanIf_00773]
CanIfTxPduReadNotifyStatus	1	[ECUC_CanIf_00589]
CanIfTxPduTriggerTransmit	0..1	[ECUC_CanIf_00840]
CanIfTxPduTruncation	1	[ECUC_CanIf_00845]
CanIfTxPduType	1	[ECUC_CanIf_00593]
CanIfTxPduBufferRef	1	[ECUC_CanIf_00831]
CanIfTxPduRef	1	[ECUC_CanIf_00603]

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfTxPduXLParams	0..1	CAN XL parameters. Identifies the CanIfTxPduCfg as CAN XL PDU.

⌋

[ECUC_CanIf_00592] Definition of EcucIntegerParamDef CanIfTxPduCanId ⌈

Parameter Name	CanIfTxPduCanId	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	[20, CP_TPS_SystemTemplate]	
Description	CAN Identifier of transmit CAN L-PDUs used by the CAN Driver for CAN L-PDU transmission. Range: ExtId (bit 31), CanFd (bit 30), and 11 Bit For Standard CAN Identifier ... 29 Bit For Extended CAN identifier The CAN Identifier may be omitted for dynamic transmit L-PDUs and CAN XL PDUs. Dependency: Not relevant if CanIfTxPduXLParams is configured and CanIfXLSduType is not 3.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 3758096383	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00823] Definition of EcucIntegerParamDef CanIfTxPduCanIdMask ⌈

Parameter Name	CanIfTxPduCanIdMask	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	[20, CP_TPS_SystemTemplate]	
Description	Identifier mask which denotes relevant bits in the CAN Identifier. This parameter may be used to keep parts of the CAN Identifier of dynamic transmit L-PDUs static. Range: ExtId (bit 31), CanFd (bit 30), and 11 bits for Standard CAN Identifier, 29 bits for Extended CAN Identifier.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 3758096383	
Default value	3758096383	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00591] Definition of EcucIntegerParamDef CanIfTxPduId ⌈

Parameter Name	CanIfTxPduId	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	ECU wide unique, symbolic handle for transmit CAN L-SDU. Range: 0..max. number of CantTxPduIds	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 4294967295	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_CanIf_00773] Definition of EcucBooleanParamDef CanIfTxPduPnFilterPdu ⌈

Parameter Name	CanIfTxPduPnFilterPdu	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	If CanIfPublicPnFilterSupport is enabled, by this parameter PDUs could be configured which will pass the CanIfPnFilter. If there is no CanIfTxPduPnFilterPdu configured per controller, the corresponding controller applies no CanIfPnFilter. Dependency: This parameter shall only be configurable if CanIfPublicPnSupport equals True.	
Multiplicity	0..1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00589] Definition of EcucBooleanParamDef CanIfTxPduReadNotifyStatus ⌈

Parameter Name	CanIfTxPduReadNotifyStatus	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	Enables and disables transmit confirmation for each transmit CAN L-SDU for reading its notification status. True: Enabled False: Disabled Dependency: CanIfPublicReadTxPduNotifyStatusApi must be enabled.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00840] Definition of EcucBooleanParamDef CanIfTxPduTriggerTransmit ⌈

Parameter Name	CanIfTxPduTriggerTransmit	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	Determines if or if not CanIf shall use the trigger transmit API for this PDU.	
Multiplicity	0..1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00845] Definition of EcucBooleanParamDef CanIfTxPduTruncation ⌈

Parameter Name	CanIfTxPduTruncation	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	Enables/disables truncation of PDUs that exceed the configured size.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	true	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00593] Definition of EcucEnumerationParamDef CanIfTxPduType ⌈

Parameter Name	CanIfTxPduType	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	Defines the type of each transmit CAN L-PDU.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	DYNAMIC	CAN ID is defined at runtime.
	STATIC	CAN ID is defined at compile-time.
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00831] Definition of EcucReferenceDef CanIfTxPduBufferRef ⌈

Parameter Name	CanIfTxPduBufferRef	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	Configurable reference to a CanIf buffer configuration.	
Multiplicity	1	
Type	Reference to CanIfBufferCfg	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00603] Definition of EcucReferenceDef CanIfTxPduRef ⌈

Parameter Name	CanIfTxPduRef	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

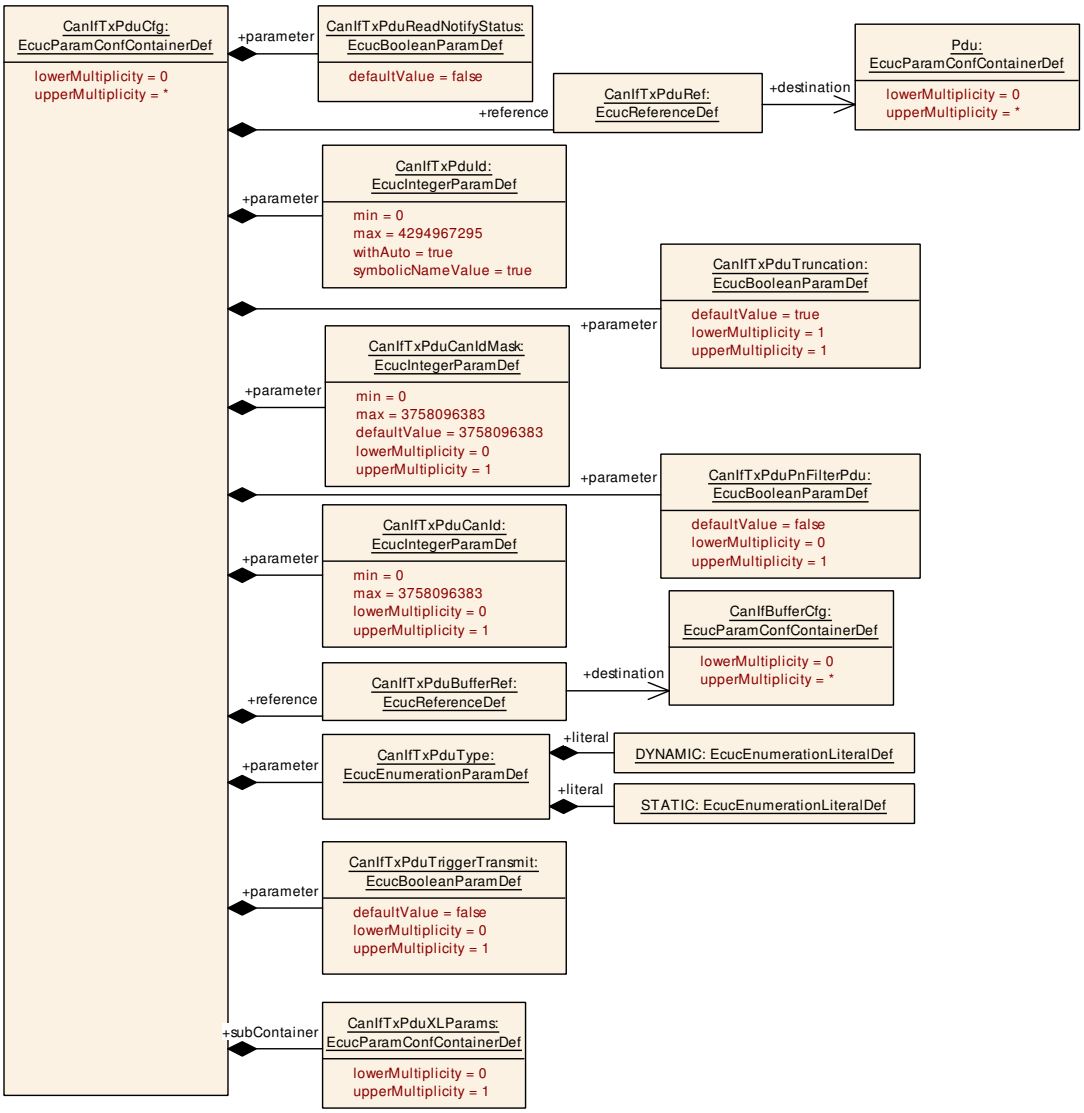


Figure 10.7: AR_EcucDef_CanIfTxPduCfg

10.1.6 CanIfTxPduXLParams

[ECUC_CanIf_00855] Definition of EcucParamConfContainerDef CanIfTxPduXL-Params ⌈

Container Name	CanIfTxPduXLParams	
Parent Container	CanIfTxPduCfg	
Upstream mapping defined	–	
Description	CAN XL parameters. Identifies the CanIfTxPduCfg as CAN XL PDU.	
Multiplicity	0..1	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfXLAcceptanceField	0..1	[ECUC_CanIf_00859]
CanIfXLPriorityId	0..1	[ECUC_CanIf_00856]
CanIfXLSduType	0..1	[ECUC_CanIf_00858]
CanIfXLVcid	0..1	[ECUC_CanIf_00857]

No Included Containers

⌋

For parameter table [ECUC_CanIf_00859] CanIfXLAcceptanceField, see definition below container CanIfRxPduXLParams.

For parameter table [ECUC_CanIf_00856] CanIfXLPriorityId, see definition below container CanIfRxPduXLParams.

For parameter table [ECUC_CanIf_00858] CanIfXLSduType, see definition below container CanIfRxPduXLParams.

For parameter table [ECUC_CanIf_00857] CanIfXLVcid, see definition below container CanIfRxPduXLParams.

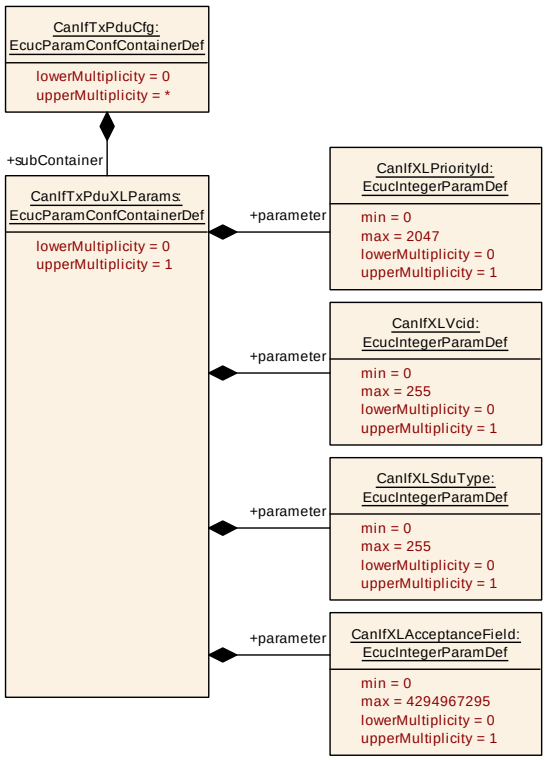


Figure 10.8: AR_EcucDef_CanIfTxPduXLParams

10.1.7 CanIfRxPduCfg

[ECUC_CanIf_00249] Definition of EcucParamConfContainerDef CanIfRxPduCfg ⌈

Container Name	CanIfRxPduCfg	
Parent Container	CanIfInitCfg	
Upstream mapping defined	–	
Description	This container contains the configuration (parameters) of each receive CAN L-PDU. The SHORT-NAME of "CanIfRxPduConfig" container itself represents the symolic name of Receive L-PDU. This L-SDU produces a meta data item of type CAN_ID_32 and TIMETUPLE_TYPE_PTR, and, depending on CanIfRxPduXLParams, meta data items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfRxPduCanId	0..1	[ECUC_CanIf_00598]
CanIfRxPduCanIdMask	0..1	[ECUC_CanIf_00822]
CanIfRxPduDataLength	1	[ECUC_CanIf_00599]
CanIfRxPduDataLengthCheck	1	[ECUC_CanIf_00846]
CanIfRxPduId	1	[ECUC_CanIf_00597]
CanIfRxPduReadData	1	[ECUC_CanIf_00600]
CanIfRxPduReadNotifyStatus	1	[ECUC_CanIf_00595]
CanIfRxPduHrhIdRef	1	[ECUC_CanIf_00602]
CanIfRxPduRef	1	[ECUC_CanIf_00601]

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfRxPduCanIdRange	0..1	Optional container that allows to map a range of CAN Ids to one PduId.
CanIfRxPduXLParams	0..1	CAN XL parameters. Identifies the CanIfRxPduCfg as CAN XL PDU.

⌋

[ECUC_CanIf_00598] Definition of EcucIntegerParamDef CanIfRxPduCanId ⌈

Parameter Name	CanIfRxPduCanId
Parent Container	CanIfRxPduCfg
Upstream mapping defined	[20, CP_TPS_SystemTemplate]
Description	CAN Identifier of Receive CAN L-PDUs used by the CAN Interface. Exa: Software Filtering. This parameter is used if exactly one Can Identifier is assigned to the Pdu. If a range is assigned then the CanIfRxPduCanIdRange parameter shall be used. Range: ExtId (bit 31), CanFd (bit 30), and 11 Bit For Standard CAN Identifier ... 29 Bit For Extended CAN identifier
Multiplicity	0..1

▽

△

Type	EcucIntegerParamDef	
Range	0 .. 3758096383	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00822] Definition of EcucIntegerParamDef CanIfRxPduCanIdMask

⌈

Parameter Name	CanIfRxPduCanIdMask	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	[20, CP_TPS_SystemTemplate]	
Description	Identifier mask which denotes relevant bits in the CAN Identifier. This parameter defines a CAN Identifier range in an alternative way to CanIfRxPduCanIdRange. It identifies the bits of the configured CAN Identifier that must match the received CAN Identifier. Range: ExtId (bit 31), CanFd (bit 30), and 11 bits for Standard CAN Identifier, 29 bits for Extended CAN Identifier.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 3758096383	
Default value	536870911	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00599] Definition of EcucIntegerParamDef CanIfRxPduDataLength

⌈

Parameter Name	CanIfRxPduDataLength	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	[20, CP_TPS_SystemTemplate]	
Description	Data length of the received CAN L-PDUs used by the CAN Interface. This information is used for Data Length Check. Additionally it might specify the valid bits in case of the discrete DLC for CAN FD L-PDUs > 8 bytes. The data area size of a CAN L-PDU can have a range from 0 to 2048 bytes. Dependency: If CanIfRxPduDataLength > 8 then bit 30 of CanIfRxPduCanId must be 1. If CanIfRxPduDataLength > 64 then CanIfRxPduXLParams must be present.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 2048	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00846] Definition of EcucBooleanParamDef CanIfRxPduDataLengthCheck ⌈

Parameter Name	CanIfRxPduDataLengthCheck	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	This parameter switches the message specific data length check. True: Data length check will be executed during the reception of this PDU. False: No data length check will be executed during the reception of this PDU.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	true	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00597] Definition of EcucIntegerParamDef CanIfRxPduId ⌈

Parameter Name	CanIfRxPduId	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	ECU wide unique, symbolic handle for receive CAN L-SDU. It shall fulfill ANSI/ AUTOSAR definitions for constant defines. Range: 0..max. number of defined CanRxPduIds Dependency: Not relevant if CanIfRxPduXLParams is configured and CanIfXLSduType is not 3.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 4294967295	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_CanIf_00600] Definition of EcucBooleanParamDef CanIfRxPduReadData ⌈

Parameter Name	CanIfRxPduReadData	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	Enables and disables the Rx buffering for reading of received L-SDU data. True: Enabled False: Disabled	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00595] Definition of EcucBooleanParamDef CanIfRxPduReadNotifyStatus ⌈

Parameter Name	CanIfRxPduReadNotifyStatus	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	Enables and disables receive indication for each receive CAN L-SDU for reading its notification status. True: Enabled False: Disabled Dependency: CanIfPublicReadRxPduNotifyStatusApi must be enabled.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	false	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00602] Definition of EcucReferenceDef CanIfRxPduHrhIdRef ⌈

Parameter Name	CanIfRxPduHrhIdRef	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	The HRH to which Rx L-PDU belongs to, is referred through this parameter. Dependency: This information has to be derived from the CAN Driver configuration.	
Multiplicity	1	
Type	Reference to CanIfHrhCfg	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00601] Definition of EcucReferenceDef CanIfRxPduRef ⌈

Parameter Name	CanIfRxPduRef	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.	
Multiplicity	1	
Type	Reference to Pdu	
Post-Build Value	Loadable	Selectable

⌋

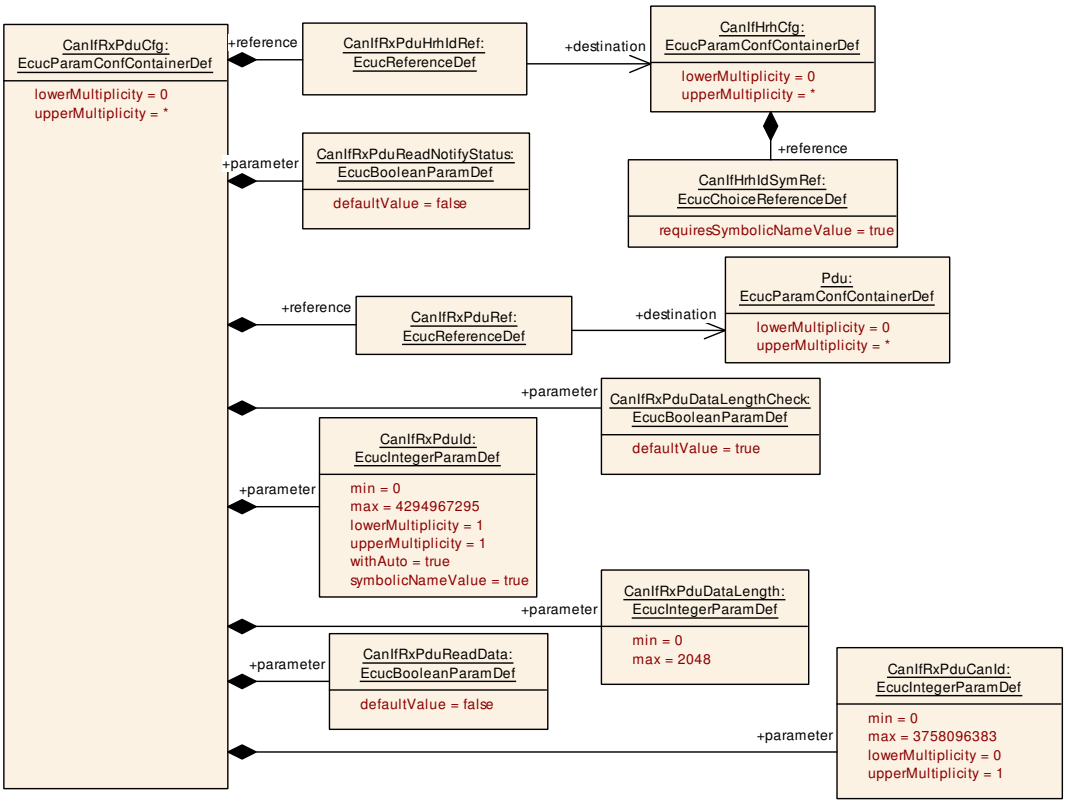


Figure 10.9: AR_EcucDef_CanIfRxPduCfg

10.1.8 CanIfRxPduXLParams

[ECUC_CanIf_00860] Definition of EcucParamConfContainerDef CanIfRxPduXL-Params ⌈

Container Name	CanIfRxPduXLParams	
Parent Container	CanIfRxPduCfg	
Upstream mapping defined	–	
Description	CAN XL parameters. Identifies the CanIfRxPduCfg as CAN XL PDU.	
Multiplicity	0..1	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfXLAcceptanceField	0..1	[ECUC_CanIf_00859]
CanIfXLPriorityId	0..1	[ECUC_CanIf_00856]
CanIfXLSduType	0..1	[ECUC_CanIf_00858]
CanIfXLVcid	0..1	[ECUC_CanIf_00857]

No Included Containers

⌋

[ECUC_CanIf_00859] Definition of EcucIntegerParamDef CanIfXLAcceptanceField ⌈

Parameter Name	CanIfXLAcceptanceField	
Parent Container	CanIfRxPduXLParams, CanIfTxPduXLParams	
Upstream mapping defined	–	
Description	Acceptance field of a CAN XL message. Dependency: Not relevant for CanIfXLSduType = 3.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00856] Definition of EcucIntegerParamDef CanIfXLPriorityId ⌈

Parameter Name	CanIfXLPriorityId	
Parent Container	CanIfRxPduXLParams, CanIfTxPduXLParams	
Upstream mapping defined	–	
Description	Priority ID of a CAN XL message.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 2047	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00858] Definition of EcucIntegerParamDef CanIfXLSduType ⌈

Parameter Name	CanIfXLSduType	
Parent Container	CanIfRxPduXLParams, CanIfTxPduXLParams	
Upstream mapping defined	–	
Description	SDU type of a CAN XL message.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00857] Definition of EcucIntegerParamDef CanIfXLVcid ⌈

Parameter Name	CanIfXLVcid	
Parent Container	CanIfRxPduXLParams, CanIfTxPduXLParams	
Upstream mapping defined	–	
Description	Virtual CAN network ID of a CAN XL message.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

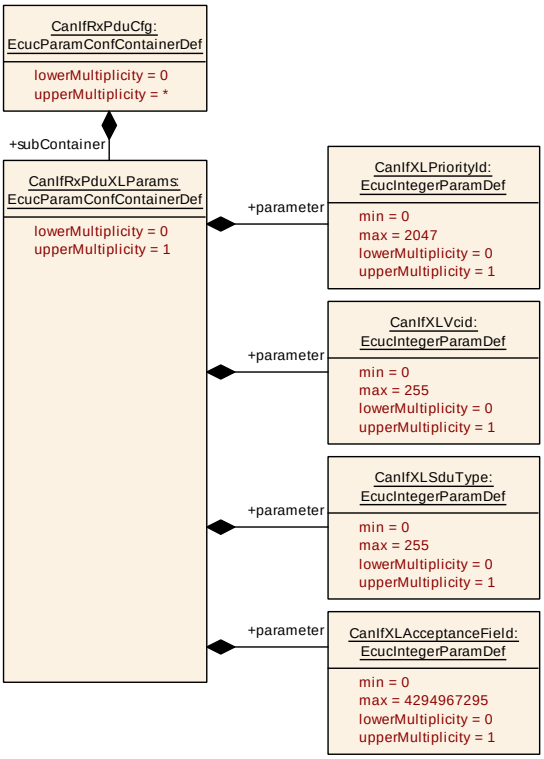


Figure 10.10: AR_EcucDef_CanIfRxPduXLParams

10.1.9 CanIfRxPduCanIdRange

[ECUC_CanIf_00743] Definition of EcucParamConfContainerDef CanIfRxPduCanIdRange ⌈

Container Name	CanIfRxPduCanIdRange
Parent Container	CanIfRxPduCfg
Upstream mapping defined	[20, CP_TPS_SystemTemplate]
Description	Optional container that allows to map a range of CAN Ids to one PduId.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfRxPduCanIdRangeLowerCanId	1	[ECUC_CanIf_00745]
CanIfRxPduCanIdRangeUpperCanId	1	[ECUC_CanIf_00744]

No Included Containers

⌋

[ECUC_CanIf_00745] Definition of EcucIntegerParamDef CanIfRxPduCanId RangeLowerCanId ⌈

Parameter Name	CanIfRxPduCanIdRangeLowerCanId	
Parent Container	CanIfRxPduCanIdRange	
Upstream mapping defined	[20, CP_TPS_SystemTemplate]	
Description	Lower CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids are mapped to one PduId.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 536870911	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00744] Definition of EcucIntegerParamDef CanIfRxPduCanId RangeUpperCanId ⌈

Parameter Name	CanIfRxPduCanIdRangeUpperCanId	
Parent Container	CanIfRxPduCanIdRange	
Upstream mapping defined	[20, CP_TPS_SystemTemplate]	
Description	Upper CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids are mapped to one PduId.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 536870911	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

10.1.10 CanIfDispatchCfg

[ECUC_CanIf_00250] Definition of EcucParamConfContainerDef CanIfDispatchCfg ⌈

Container Name	CanIfDispatchCfg
Parent Container	CanIf
Upstream mapping defined	–
Description	Callback functions provided by upper layer modules of the CanIf. The callback functions defined in this container are common to all configured CAN Driver / CAN Transceiver Driver modules.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfDispatchUserCheckTrcvWakeFlagIndicationName	0..1	[ECUC_CanIf_00791]
CanIfDispatchUserCheckTrcvWakeFlagIndicationUL	0..1	[ECUC_CanIf_00792]
CanIfDispatchUserClearTrcvWufFlagIndicationName	0..1	[ECUC_CanIf_00789]
CanIfDispatchUserClearTrcvWufFlagIndicationUL	0..1	[ECUC_CanIf_00790]
CanIfDispatchUserConfirmCtrlPnAvailabilityName	0..1	[ECUC_CanIf_00867]
CanIfDispatchUserConfirmCtrlPnAvailabilityUL	0..1	[ECUC_CanIf_00868]
CanIfDispatchUserConfirmPnAvailabilityName	0..1	[ECUC_CanIf_00819]
CanIfDispatchUserConfirmPnAvailabilityUL	0..1	[ECUC_CanIf_00820]
CanIfDispatchUserCtrlBusOffName	0..1	[ECUC_CanIf_00525]
CanIfDispatchUserCtrlBusOffUL	1	[ECUC_CanIf_00547]
CanIfDispatchUserCtrlModeIndicationName	0..1	[ECUC_CanIf_00683]
CanIfDispatchUserCtrlModeIndicationUL	1	[ECUC_CanIf_00684]
CanIfDispatchUserTrcvModeIndicationName	0..1	[ECUC_CanIf_00685]
CanIfDispatchUserTrcvModeIndicationUL	0..1	[ECUC_CanIf_00686]
CanIfDispatchUserValidateWakeupEventName	0..1	[ECUC_CanIf_00531]
CanIfDispatchUserValidateWakeupEventUL	0..1	[ECUC_CanIf_00549]

No Included Containers

⌋

[ECUC_CanIf_00791] Definition of EcucFunctionNameDef CanIfDispatchUserCheckTrcvWakeFlagIndicationName ⌈

Parameter Name	CanIfDispatchUserCheckTrcvWakeFlagIndicationName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–

▽

△

Description	This parameter defines the name of <User_CheckTrcvWakeFlagIndication>. If CanIf DispatchUserCheckTrcvWakeFlagIndicationUL equals CAN_SM the name of <User_ CheckTrcvWakeFlagIndication> is fixed. If it equals CDD, the name is selectable. If Can IfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True, this parameter shall be configurable. Dependency: CanIfDispatchUserCheckTrcvWakeFlagIndicationUL, CanIfPublicPn Support, CanIfPublicTrcvPnEnable
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00792] Definition of EcucEnumerationParamDef CanIfDispatch UserCheckTrcvWakeFlagIndicationUL ⌈

Parameter Name	CanIfDispatchUserCheckTrcvWakeFlagIndicationUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer module to which the CheckTrcvWakeFlag Indication from the Driver modules have to be routed. If CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True, this parameter shall be configurable. Dependency: CanIfPublicPnSupport, CanIfPublicTrcvPnEnable	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CDD	Complex Driver
Pre-Compile only	true	

⌋

[ECUC_CanIf_00789] Definition of EcucFunctionNameDef CanIfDispatchUser ClearTrcvWufFlagIndicationName ⌈

Parameter Name	CanIfDispatchUserClearTrcvWufFlagIndicationName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_ClearTrcvWufFlagIndication>. If CanIf DispatchUserClearTrcvWufFlagIndicationUL equals CAN_SM the name of <User_ ClearTrcvWufFlagIndication> is fixed. If it equals CDD, the name is selectable. If CanIf PublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True, this parameter shall be configurable. Dependency: CanIfDispatchUserClearTrcvWufFlagIndicationUL, CanIfPublicPn Support, CanIfPublicTrcvPnEnable
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–

▽

△

Pre-Compile only	true

⌋

[ECUC_CanIf_00790] Definition of EcucEnumerationParamDef CanIfDispatchUserClearTrcvWufFlagIndicationUL ⌈

Parameter Name	CanIfDispatchUserClearTrcvWufFlagIndicationUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer module to which the ClearTrcvWufFlag Indication from the Driver modules have to be routed. If CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True, this parameter shall be configurable. Dependency: CanIfPublicPnSupport, CanIfPublicTrcvPnEnable	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CDD	Complex Driver
Pre-Compile only	true	

⌋

[ECUC_CanIf_00867] Definition of EcucFunctionNameDef CanIfDispatchUserConfirmCtrlPnAvailabilityName ⌈

Parameter Name	CanIfDispatchUserConfirmCtrlPnAvailabilityName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_ConfirmCtrlPnAvailability>. If CanIf DispatchUserConfirmCtrlPnAvailabilityUL equals CAN_SM the name of <User_Confirm CtrlPnAvailability> is fixed. If it equals CDD, the name is selectable. If CanIfPublicPn Support equals True and CanIfPublicCtrlPnEnable equals True, this parameter shall be configurable. Dependency: CanIfDispatchUserConfirmCtrlPnAvailabilityUL,CanIfPublicPnSupport, CanIfPublicCtrlPnEnable
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00868] Definition of EcucEnumerationParamDef CanIfDispatchUserConfirmCtrlPnAvailabilityUL ⌈

Parameter Name	CanIfDispatchUserConfirmCtrlPnAvailabilityUL
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–

▽

△

Description	This parameter defines the upper layer module to which the ConfirmCtrlPnAvailability notification from the Driver modules have to be routed. If CanIfPublicPnSupport equals True and CanIfPublicCtrlPnEnable equals True, this parameter shall be configurable. Dependency: CanIfPublicPnSupport, CanIfPublicCtrlPnEnable	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CDD	Complex Driver
Pre-Compile only	true	

⌋

[ECUC_CanIf_00819] Definition of EcucFunctionNameDef CanIfDispatchUserConfirmPnAvailabilityName ⌈

Parameter Name	CanIfDispatchUserConfirmPnAvailabilityName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_ConfirmPnAvailability>. If CanIfDispatch UserConfirmPnAvailabilityUL equals CAN_SM the name of <User_ConfirmPn Availability> is fixed. If it equals CDD, the name is selectable. If CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True, this parameter shall be configurable. Dependency: CanIfDispatchUserConfirmPnAvailabilityUL, CanIfPublicPnSupport, Can IfPublicTrcvPnEnable
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00820] Definition of EcucEnumerationParamDef CanIfDispatch UserConfirmPnAvailabilityUL ⌈

Parameter Name	CanIfDispatchUserConfirmPnAvailabilityUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer module to which the ConfirmPnAvailability notification from the Driver modules have to be routed. If CanIfPublicPnSupport equals True and CanIfPublicTrcvPnEnable equals True, this parameter shall be configurable. Dependency: CanIfPublicPnSupport, CanIfPublicTrcvPnEnable	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CDD	Complex Driver
Pre-Compile only	true	

⌋

[ECUC_CanIf_00525] Definition of EcucFunctionNameDef CanIfDispatchUserCtrlBusOffName ⌈

Parameter Name	CanIfDispatchUserCtrlBusOffName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_ControllerBusOff>. This parameter depends on the parameter CanIfDispatchUserCtrlBusOffUL. If CanIfDispatchUserCtrlBusOffUL equals CAN_SM the name of <User_ControllerBusOff> is fixed. If CanIfDispatchUserCtrlBusOffUL equals CDD, the name of <User_ControllerBusOff> is selectable. Dependency: CanIfDispatchUserCtrlBusOffUL
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00547] Definition of EcucEnumerationParamDef CanIfDispatchUserCtrlBusOffUL ⌈

Parameter Name	CanIfDispatchUserCtrlBusOffUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer (UL) module to which the notifications of all ControllerBusOff events from the CAN Driver modules have to be routed via <User_ControllerBusOff>. There is no possibility to configure no upper layer (UL) module as the provider of <User_ControllerBusOff>.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CDD	Complex Driver
Pre-Compile only	true	

⌋

[ECUC_CanIf_00683] Definition of EcucFunctionNameDef CanIfDispatchUserCtrlModeIndicationName ⌈

Parameter Name	CanIfDispatchUserCtrlModeIndicationName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_ControllerModeIndication>. This parameter depends on the parameter CanIfDispatchUserCtrlModeIndicationUL. If CanIfDispatchUserCtrlModeIndicationUL equals CAN_SM the name of <User_ControllerModeIndication> is fixed. If CanIfDispatchUserCtrlModeIndicationUL equals CDD, the name of <User_ControllerModeIndication> is selectable. Dependency: CanIfDispatchUserCtrlModeIndicationUL
Multiplicity	0..1

▽

△

Type	EcucFunctionNameDef
Default value	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00684] Definition of EcucEnumerationParamDef CanIfDispatchUserCtrlModeIndicationUL ⌈

Parameter Name	CanIfDispatchUserCtrlModeIndicationUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer (UL) module to which the notifications of all ControllerTransition events from the CAN Driver modules have to be routed via <User_ControllerModeIndication>.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CAN_SM_AND_MKA	CAN State Manager and MKA
	CDD	Complex Driver
	CDD_AND_MKA	Complex Driver and MKA
Pre-Compile only	true	

⌋

[ECUC_CanIf_00685] Definition of EcucFunctionNameDef CanIfDispatchUserTrcvModeIndicationName ⌈

Parameter Name	CanIfDispatchUserTrcvModeIndicationName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_TrcvModeIndication>. This parameter depends on the parameter CanIfDispatchUserTrcvModeIndicationUL. If CanIfDispatchUserTrcvModeIndicationUL equals CAN_SM the name of <User_TrcvModeIndication> is fixed. If CanIfDispatchUserTrcvModeIndicationUL equals CDD, the name of <User_TrcvModeIndication> is selectable. Dependency: CanIfDispatchUserTrcvModeIndicationUL
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00686] Definition of EcucEnumerationParamDef CanIfDispatchUserTrcvModeIndicationUL ⌈

Parameter Name	CanIfDispatchUserTrcvModeIndicationUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer (UL) module to which the notifications of all TransceiverTransition events from the CAN Transceiver Driver modules have to be routed via <User_TrcvModeIndication>. If no UL module is configured, no upper layer callback function will be called.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CAN_SM	CAN State Manager
	CDD	Complex Driver
Pre-Compile only	true	

⌋

[ECUC_CanIf_00531] Definition of EcucFunctionNameDef CanIfDispatchUserValidateWakeupEventName ⌈

Parameter Name	CanIfDispatchUserValidateWakeupEventName
Parent Container	CanIfDispatchCfg
Upstream mapping defined	–
Description	This parameter defines the name of <User_ValidateWakeupEvent>. This parameter depends on the parameter CanIfDispatchUserValidateWakeupEventUL. If CanIf DispatchUserValidateWakeupEventUL equals ECUM, the name of <User_Validate WakeupEvent> is fixed. If CanIfDispatchUserValidateWakeupEventUL equals CDD, the name of <User_ValidateWakeupEvent> is selectable. Dependency: CanIfDispatchUserValidateWakeupEventUL
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Length	1-32
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_CanIf_00549] Definition of EcucEnumerationParamDef CanIfDispatchUserValidateWakeupEventUL ⌈

Parameter Name	CanIfDispatchUserValidateWakeupEventUL	
Parent Container	CanIfDispatchCfg	
Upstream mapping defined	–	
Description	This parameter defines the upper layer (UL) module to which the notifications about positive former requested wake up sources have to be routed via <User_Validate WakeupEvent>.	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CDD	Complex Driver

▽

△

	ECUM	ECU State Manager
Pre-Compile only	true	

⌋

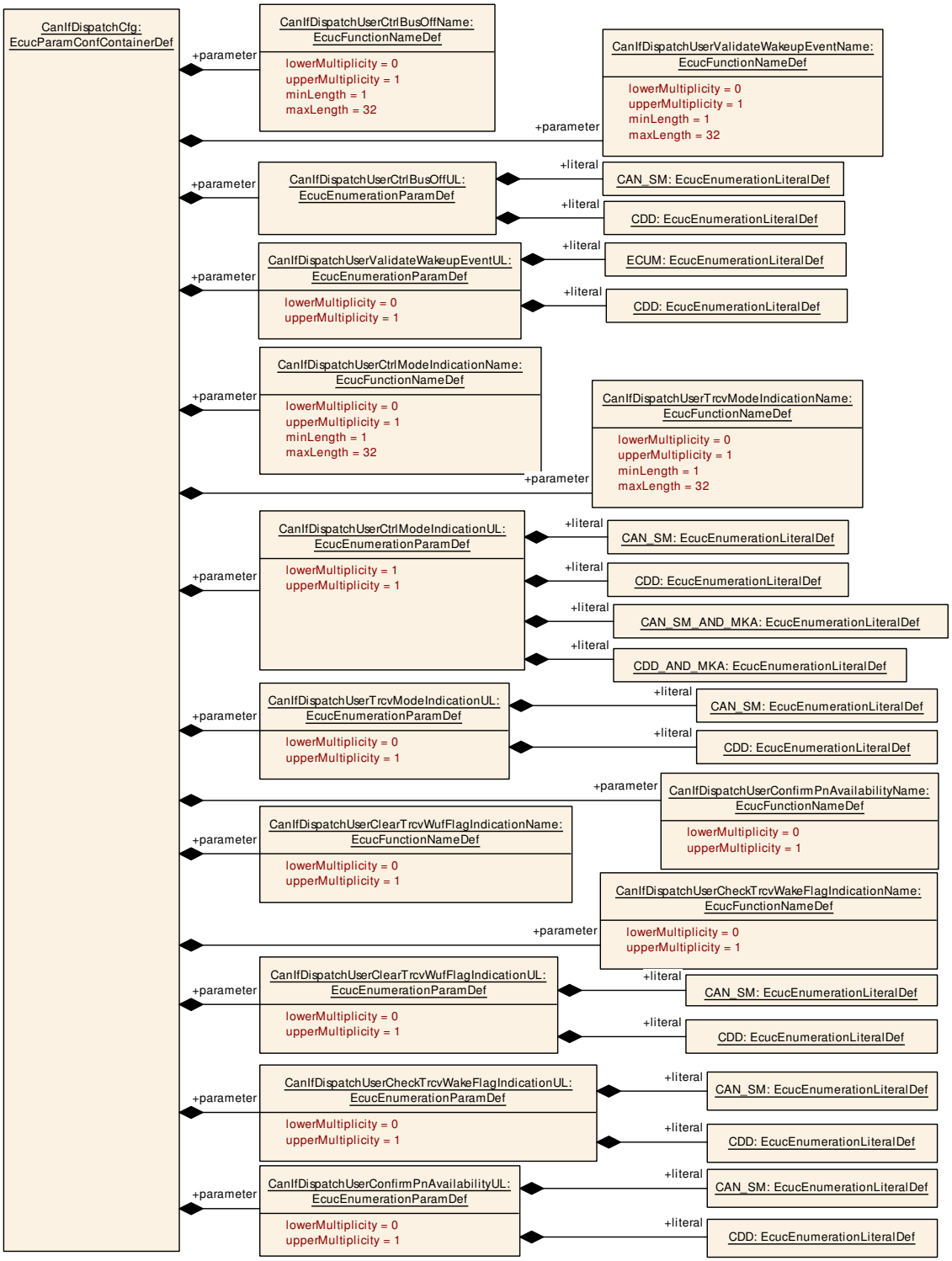


Figure 10.11: AR_EcucDef_CanIfDispatchCfg

10.1.11 CanIfCtrlCfg

[ECUC_CanIf_00546] Definition of EcucParamConfContainerDef CanIfCtrlCfg ⌈

Container Name	CanIfCtrlCfg
Parent Container	CanIfCtrlDrvCfg
Upstream mapping defined	–
Description	This container contains the configuration (parameters) of an adressed CAN controller by an underlying CAN Driver module. This container is configurable per CAN controller.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfCtrlId	1	[ECUC_CanIf_00647]
CanIfCtrlWakeupSupport	1	[ECUC_CanIf_00637]
CanIfCtrlCanCtrlRef	1	[ECUC_CanIf_00636]

No Included Containers

⌋

[ECUC_CanIf_00647] Definition of EcucIntegerParamDef CanIfCtrlId ⌈

Parameter Name	CanIfCtrlId
Parent Container	CanIfCtrlCfg
Upstream mapping defined	–
Description	This parameter abstracts from the CAN Driver specific parameter Controller. Each controller of all connected CAN Driver modules shall be assigned to one specific ControllerId of the CanIf. Range: 0..number of configured controllers of all CAN Driver modules
Multiplicity	1
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)
Range	0 .. 255
Default value	–
With Auto	true
Pre-Compile only	true

⌋

[ECUC_CanIf_00637] Definition of EcucBooleanParamDef CanIfCtrlWakeupSupport ⌈

Parameter Name	CanIfCtrlWakeupSupport
Parent Container	CanIfCtrlCfg
Upstream mapping defined	–
Description	This parameter defines if a respective controller of the referenced CAN Driver modules is queriable for wake up events. True: Enabled False: Disabled

▽

△

Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00636] Definition of EcucReferenceDef CanIfCtrlCanCtrlRef ⌈

Parameter Name	CanIfCtrlCanCtrlRef
Parent Container	CanIfCtrlCfg
Upstream mapping defined	–
Description	This parameter references to the logical handle of the underlying CAN controller from the CAN Driver module to be served by the CAN Interface module. The following parameters of CanController config container shall be referenced by this link: Can ControllerId, CanWakeupSourceRef Range: 0..max. number of underlying supported CAN controllers Dependency: amount of CAN controllers
Multiplicity	1
Type	Symbolic name reference to CanController
Pre-Compile only	true

⌋

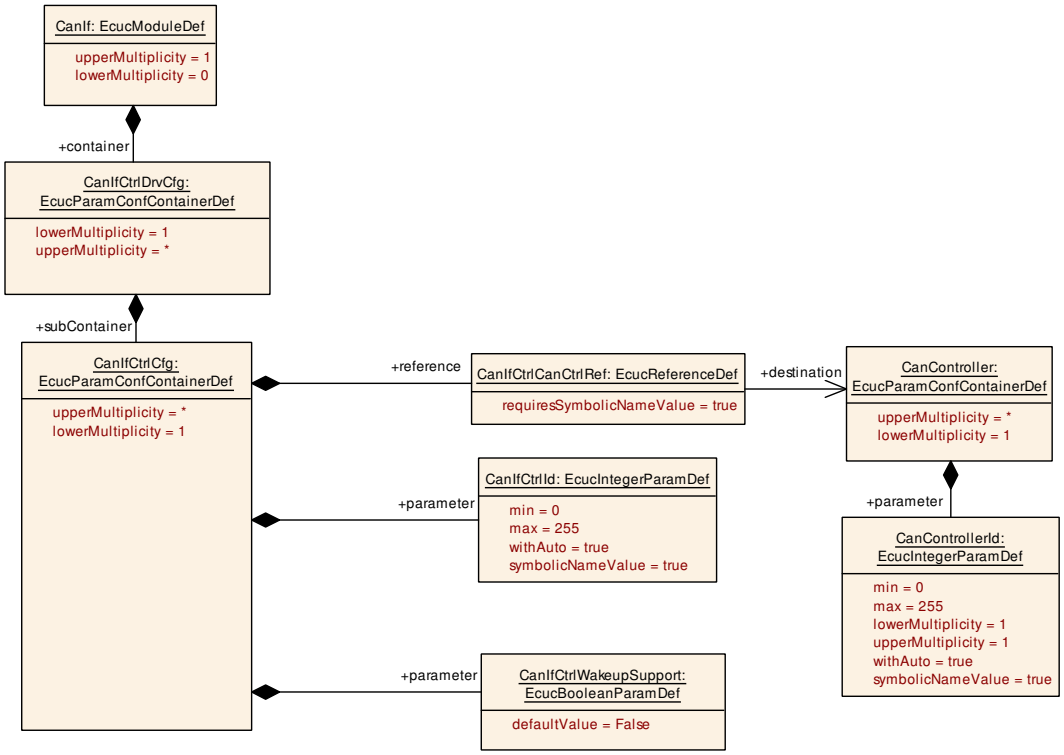


Figure 10.12: AR_EcucDef_CanIfCtrlCfg

10.1.12 CanIfCtrlDrvCfg

[ECUC_CanIf_00253] Definition of EcucParamConfContainerDef CanIfCtrlDrvCfg ⌈

Container Name	CanIfCtrlDrvCfg
Parent Container	CanIf
Upstream mapping defined	–
Description	Configuration parameters for all the underlying CAN Driver modules are aggregated under this container. For each CAN Driver module a seperate instance of this container has to be provided.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfCtrlDrvInitHohConfigRef	1	[ECUC_CanIf_00642]
CanIfCtrlDrvNameRef	1	[ECUC_CanIf_00638]

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfCtrlCfg	1..*	This container contains the configuration (parameters) of an adressed CAN controller by an underlying CAN Driver module. This container is configurable per CAN controller.

⌋

[ECUC_CanIf_00642] Definition of EcucReferenceDef CanIfCtrlDrvInitHohConfig Ref ⌈

Parameter Name	CanIfCtrlDrvInitHohConfigRef
Parent Container	CanIfCtrlDrvCfg
Upstream mapping defined	–
Description	Reference to the Init Hoh Configuration
Multiplicity	1
Type	Reference to CanIfInitHohCfg
Pre-Compile only	true

⌋

[ECUC_CanIf_00638] Definition of EcucReferenceDef CanIfCtrlDrvNameRef ⌈

Parameter Name	CanIfCtrlDrvNameRef
Parent Container	CanIfCtrlDrvCfg
Upstream mapping defined	–

▽

△

Description	CAN Interface Driver Reference. This reference can be used to get any information (Ex. Driver Name, Vendor ID) from the CAN driver. The CAN Driver name can be derived from the ShortName of the CAN driver module.
Multiplicity	1
Type	Reference to CanGeneral
Pre-Compile only	true

⌋

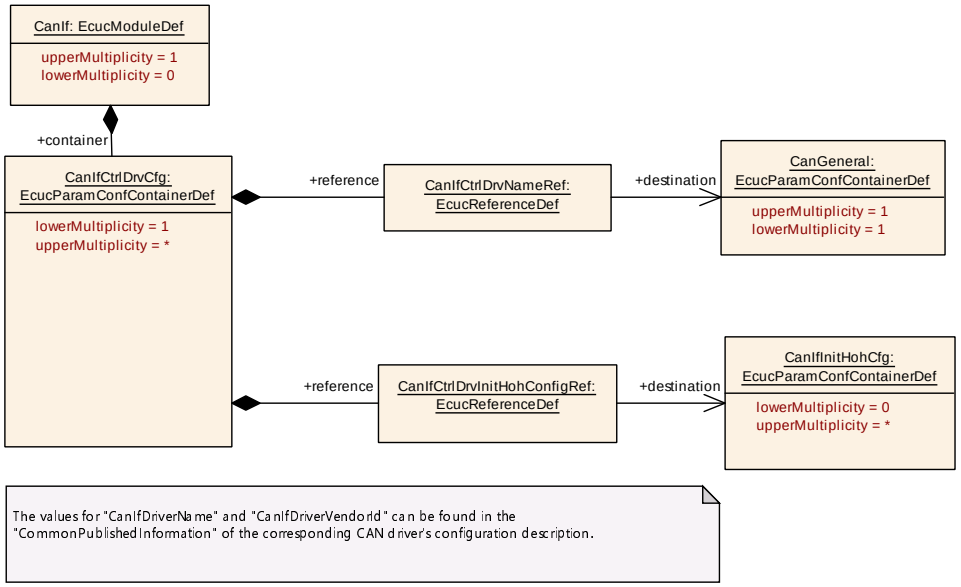


Figure 10.13: AR_EcucDef_CanIfCtrlDrvCfg

10.1.13 CanIfTrcvDrvCfg

[ECUC_CanIf_00273] Definition of EcucParamConfContainerDef CanIfTrcvDrvCfg ⌈

Container Name	CanIfTrcvDrvCfg
Parent Container	CanIf
Upstream mapping defined	–
Description	This container contains the configuration (parameters) of all addressed CAN transceivers by each underlying CAN Transceiver Driver module. For each CAN transceiver Driver a seperate instance of this container shall be provided.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfTrcvCfg	1..*	This container contains the configuration (parameters) of one addressed CAN transceiver by the underlying CAN Transceiver Driver module. For each CAN transceiver a seperate instance of this container has to be provided.

⌋

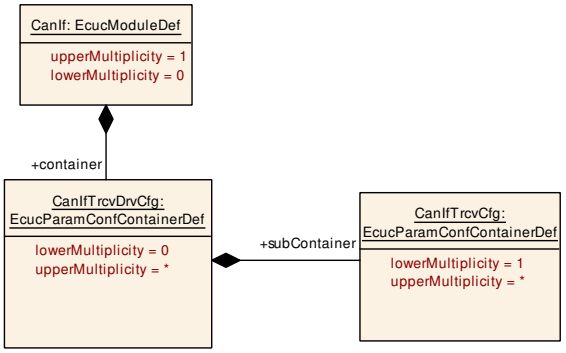


Figure 10.14: AR_EcucDef_CanIfTrcvDrvCfg

10.1.14 CanIfTrcvCfg

[ECUC_CanIf_00587] Definition of EcucParamConfContainerDef CanIfTrcvCfg ⌈

Container Name	CanIfTrcvCfg
Parent Container	CanIfTrcvDrvCfg
Upstream mapping defined	–
Description	This container contains the configuration (parameters) of one addressed CAN transceiver by the underlying CAN Transceiver Driver module. For each CAN transceiver a seperate instance of this container has to be provided.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfTrcvId	1	[ECUC_CanIf_00654]
CanIfTrcvWakeupSupport	1	[ECUC_CanIf_00606]
CanIfTrcvCanTrcvRef	1	[ECUC_CanIf_00605]

No Included Containers

⌋

[ECUC_CanIf_00654] Definition of EcucIntegerParamDef CanIfTrcvId ⌈

Parameter Name	CanIfTrcvId	
Parent Container	CanIfTrcvCfg	
Upstream mapping defined	–	
Description	This parameter abstracts from the CAN Transceiver Driver specific parameter Transceiver. Each transceiver of all connected CAN Transceiver Driver modules shall be assigned to one specific TransceiverId of the CanIf. Range: 0..number of configured transceivers of all CAN Transceiver Driver modules	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 255	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_CanIf_00606] Definition of EcucBooleanParamDef CanIfTrcvWakeupSupport ⌈

Parameter Name	CanIfTrcvWakeupSupport
Parent Container	CanIfTrcvCfg
Upstream mapping defined	–
Description	This parameter defines if a respective transceiver of the referenced CAN Transceiver Driver modules is queriable for wake up events. True: Enabled False: Disabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_CanIf_00605] Definition of EcucReferenceDef CanIfTrcvCanTrcvRef ⌈

Parameter Name	CanIfTrcvCanTrcvRef
Parent Container	CanIfTrcvCfg
Upstream mapping defined	–
Description	This parameter references to the logical handle of the underlying CAN transceiver from the CAN transceiver driver module to be served by the CAN Interface module. Range: 0..max. number of underlying supported CAN transceivers Dependency: amount of CAN transceivers
Multiplicity	1
Type	Symbolic name reference to CanTrcvChannel
Pre-Compile only	true

⌋

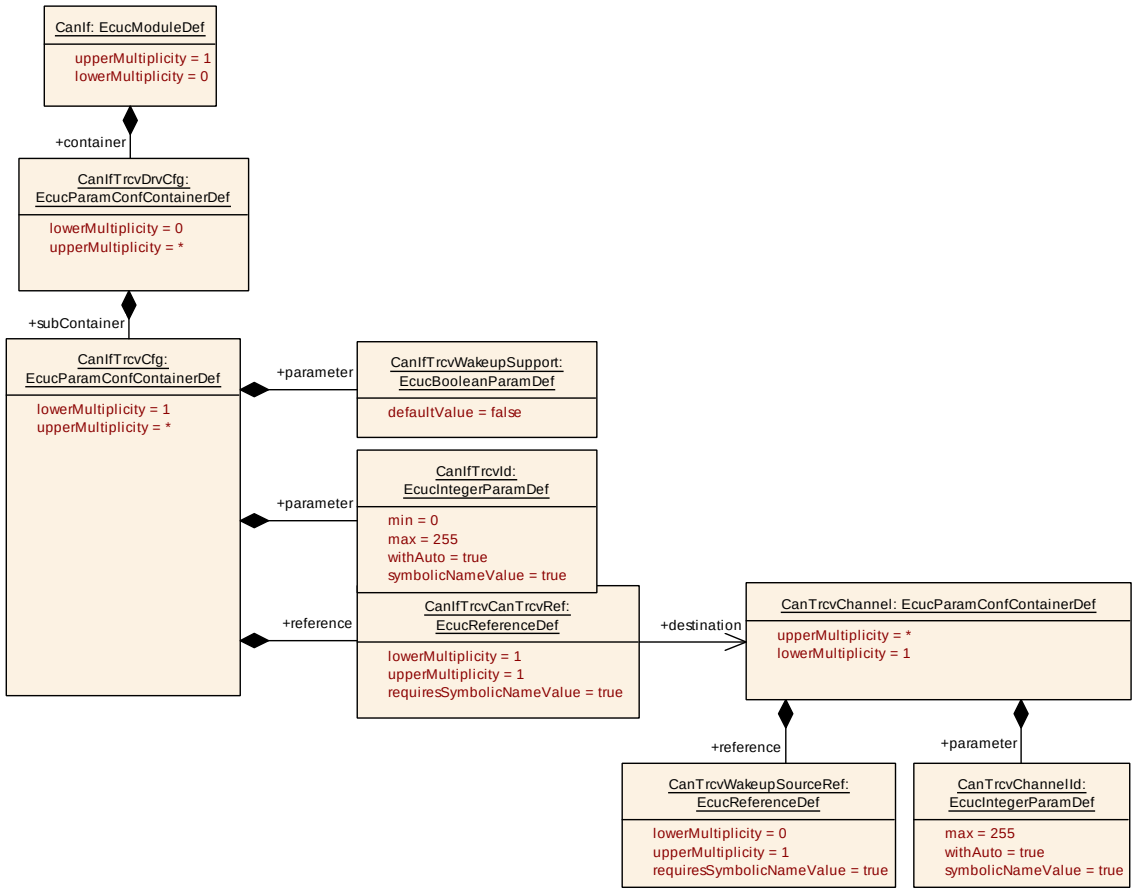


Figure 10.15: AR_EcucDef_CanIfTrcvCfg

10.1.15 CanIfInitHohCfg

[ECUC_CanIf_00257] Definition of EcucParamConfContainerDef CanIfInitHohCfg ⌈

Container Name	CanIfInitHohCfg
Parent Container	CanIfInitCfg
Upstream mapping defined	–
Description	This container contains the references to the configuration setup of each underlying CAN Driver.
Multiplicity	0..*
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
CanIfHrhCfg	0..*	This container contains configuration parameters for each hardware receive object (HRH).
CanIfHthCfg	0..*	This container contains parameters related to each HTH.

⌋

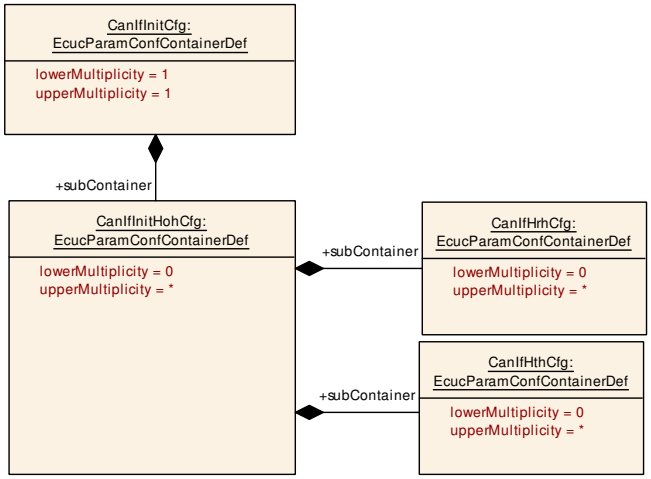


Figure 10.16: AR_EcucDef_CanIfInitHohCfg

10.1.16 CanIfHthCfg

[ECUC_CanIf_00258] Definition of EcucParamConfContainerDef CanIfHthCfg ⌈

Container Name	CanIfHthCfg	
Parent Container	CanIfInitHohCfg	
Upstream mapping defined	–	
Description	This container contains parameters related to each HTH.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfHthCanCtrlIdRef	1	[ECUC_CanIf_00625]
CanIfHthIdSymRef	1	[ECUC_CanIf_00627]

No Included Containers

⌋

[ECUC_CanIf_00625] Definition of EcucReferenceDef CanIfHthCanCtrlIdRef ⌈

Parameter Name	CanIfHthCanCtrlIdRef
Parent Container	CanIfHthCfg
Upstream mapping defined	–
Description	Reference to controller Id to which the HTH belongs to. A controller can contain one or more HTHs.

▽

△

Multiplicity	1	
Type	Reference to CanIfCtrlCfg	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00627] Definition of EcucChoiceReferenceDef CanIfHthIdSymRef

⌈

Parameter Name	CanIfHthIdSymRef	
Parent Container	CanIfHthCfg	
Upstream mapping defined	–	
Description	The parameter refers to a particular HTH object in the CanDrv configuration (see Can HardwareObject or CanXLHardwareObject). CanIf receives the following information of the CanDrv module by this reference: • CanHandleType (only for CAN 2.0 and CAN FD) • CanObjectId	
Multiplicity	1	
Type	Choice symbolic name reference to [CanHardwareObject, CanXLHardwareObject]	
Post-Build Value	Loadable	Selectable

⌋

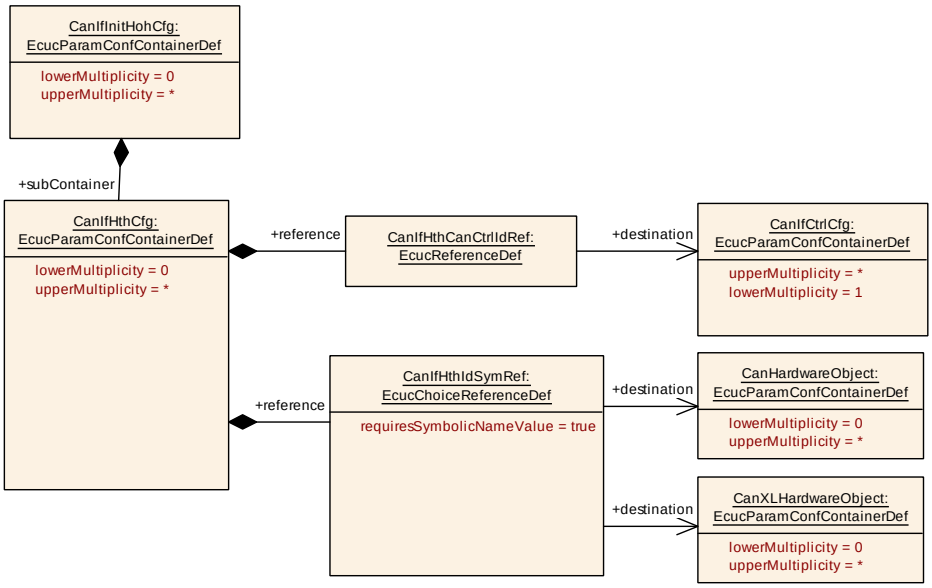


Figure 10.17: AR_EcucDef_CanIfHthCfg

10.1.17 CanIfHrhCfg

[ECUC_CanIf_00259] Definition of EcucParamConfContainerDef CanIfHrhCfg ⌈

Container Name	CanIfHrhCfg	
Parent Container	CanIfInitHohCfg	
Upstream mapping defined	–	
Description	This container contains configuration parameters for each hardware receive object (HRH).	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
CanIfHrhSoftwareFilter	1	[ECUC_CanIf_00632]
CanIfHrhCanCtrlIdRef	1	[ECUC_CanIf_00631]
CanIfHrhIdSymRef	1	[ECUC_CanIf_00634]

Included Containers		
Container Name	Multiplicity	Description
CanIfHrhRangeCfg	0..*	Defines the parameters required for configurating multiple CANID ranges for a given same HRH.

⌋

[ECUC_CanIf_00632] Definition of EcucBooleanParamDef CanIfHrhSoftwareFilter ⌈

Parameter Name	CanIfHrhSoftwareFilter
Parent Container	CanIfHrhCfg
Upstream mapping defined	–
Description	Selects the hardware receive objects by using the HRH range/list from CAN Driver configuration to define, for which HRH a software filtering has to be performed at during receive processing. True: Software filtering is enabled False: Software filtering is enabled
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true
Pre-Compile only	true

⌋

[ECUC_CanIf_00631] Definition of EcucReferenceDef CanIfHrhCanCtrlIdRef ⌈

Parameter Name	CanIfHrhCanCtrlIdRef	
Parent Container	CanIfHrhCfg	
Upstream mapping defined	–	
Description	Reference to controller Id to which the HRH belongs to. A controller can contain one or more HRHs.	
Multiplicity	1	
Type	Reference to CanIfCtrlCfg	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00634] Definition of EcucChoiceReferenceDef CanIfHrhIdSymRef ⌈

Parameter Name	CanIfHrhIdSymRef	
Parent Container	CanIfHrhCfg	
Upstream mapping defined	–	
Description	The parameter refers to a particular HRH object in the CanDrv configuration (see Can HardwareObject or CanXLHardwareObject). CanIf receives the following information of the CanDrv module by this reference: • CanHandleType (only CAN 2.0 and CAN FD) • CanObjectId	
Multiplicity	1	
Type	Choice symbolic name reference to [CanHardwareObject, CanXLHardwareObject]	
Post-Build Value	Loadable	Selectable

⌋

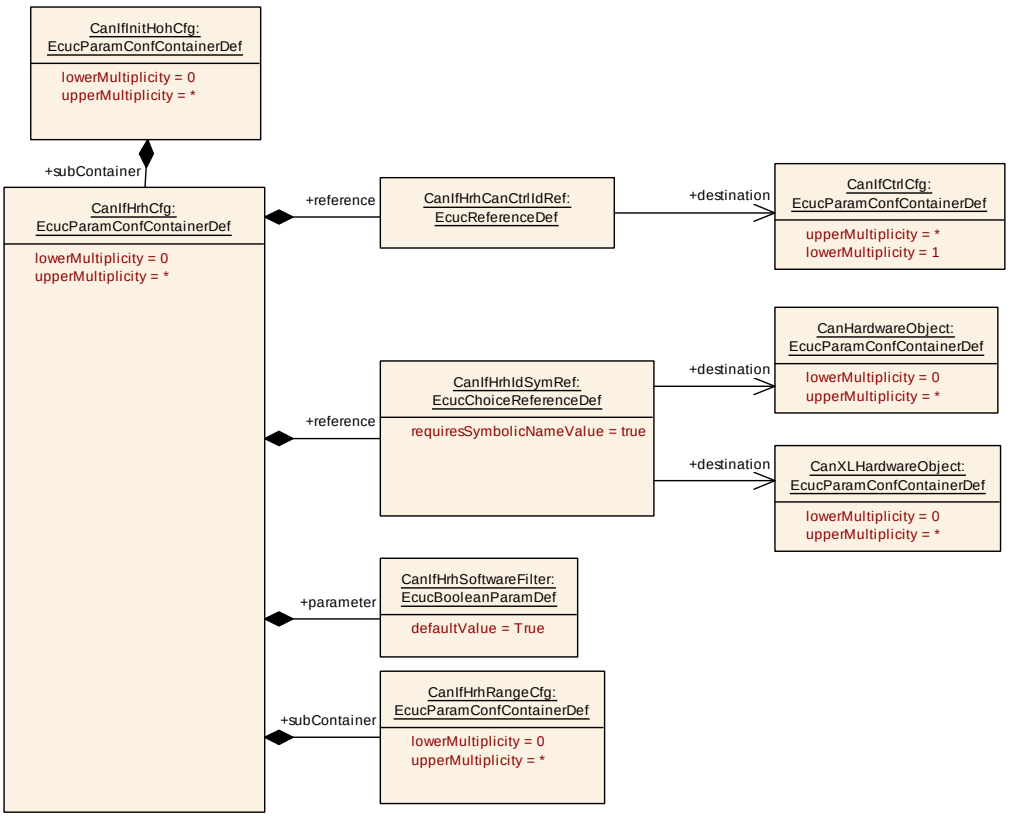


Figure 10.18: AR_EcucDef_CanIfHrhCfg

10.1.18 CanIfHrhRangeCfg

[ECUC_CanIf_00628] Definition of EcucParamConfContainerDef CanIfHrhRange Cfg ⌈

Container Name	CanIfHrhRangeCfg	
Parent Container	CanIfHrhCfg	
Upstream mapping defined	–	
Description	Defines the parameters required for configurating multiple CANID ranges for a given same HRH.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
CanIfHrhRangeBaseId	0..1	[ECUC_CanIf_00825]
CanIfHrhRangeMask	0..1	[ECUC_CanIf_00826]
CanIfHrhRangeRxPduLowerCanId	0..1	[ECUC_CanIf_00629]
CanIfHrhRangeRxPduRangeCanIdType	1	[ECUC_CanIf_00644]
CanIfHrhRangeRxPduUpperCanId	0..1	[ECUC_CanIf_00630]

No Included Containers

⌋

[ECUC_CanIf_00825] Definition of EcucIntegerParamDef CanIfHrhRangeBaseId ⌈

Parameter Name	CanIfHrhRangeBaseId	
Parent Container	CanIfHrhRangeCfg	
Upstream mapping defined	–	
Description	CAN Identifier used as base value in combination with CanIfHrhRangeMask for a masked ID range in which all CAN Ids shall pass the software filtering. The size of this parameter is limited by CanIfHrhRangeRxPduRangeCanIdType.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 536870911	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00826] Definition of EcucIntegerParamDef CanIfHrhRangeMask ⌈

Parameter Name	CanIfHrhRangeMask
Parent Container	CanIfHrhRangeCfg
Upstream mapping defined	–
Description	Used as mask value in combination with CanIfHrhRangeBaseId for a masked ID range in which all CAN Ids shall pass the software filtering. The size of this parameter is limited by CanIfHrhRangeRxPduRangeCanIdType.
Multiplicity	0..1
Type	EcucIntegerParamDef

▽

△

Range	0 .. 536870911	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00629] Definition of EcucIntegerParamDef CanIfHrhRangeRxPduLowerCanId ⌈

Parameter Name	CanIfHrhRangeRxPduLowerCanId	
Parent Container	CanIfHrhRangeCfg	
Upstream mapping defined	–	
Description	Lower CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids shall pass the software filtering.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 536870911	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00644] Definition of EcucEnumerationParamDef CanIfHrhRangeRxPduRangeCanIdType ⌈

Parameter Name	CanIfHrhRangeRxPduRangeCanIdType	
Parent Container	CanIfHrhRangeCfg	
Upstream mapping defined	–	
Description	Specifies whether a configured Range of CAN Ids shall only consider standard CAN Ids or extended CAN Ids.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	EXTENDED	All the CANIDs are of type extended only (29 bit).
	STANDARD	All the CANIDs are of type standard only (11bit).
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00630] Definition of EcucIntegerParamDef CanIfHrhRangeRxPduUpperCanId ⌈

Parameter Name	CanIfHrhRangeRxPduUpperCanId
Parent Container	CanIfHrhRangeCfg
Upstream mapping defined	–
Description	Upper CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids shall pass the software filtering.
Multiplicity	0..1

▽

△

Type	EcucIntegerParamDef	
Range	0 .. 536870911	
Default value	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

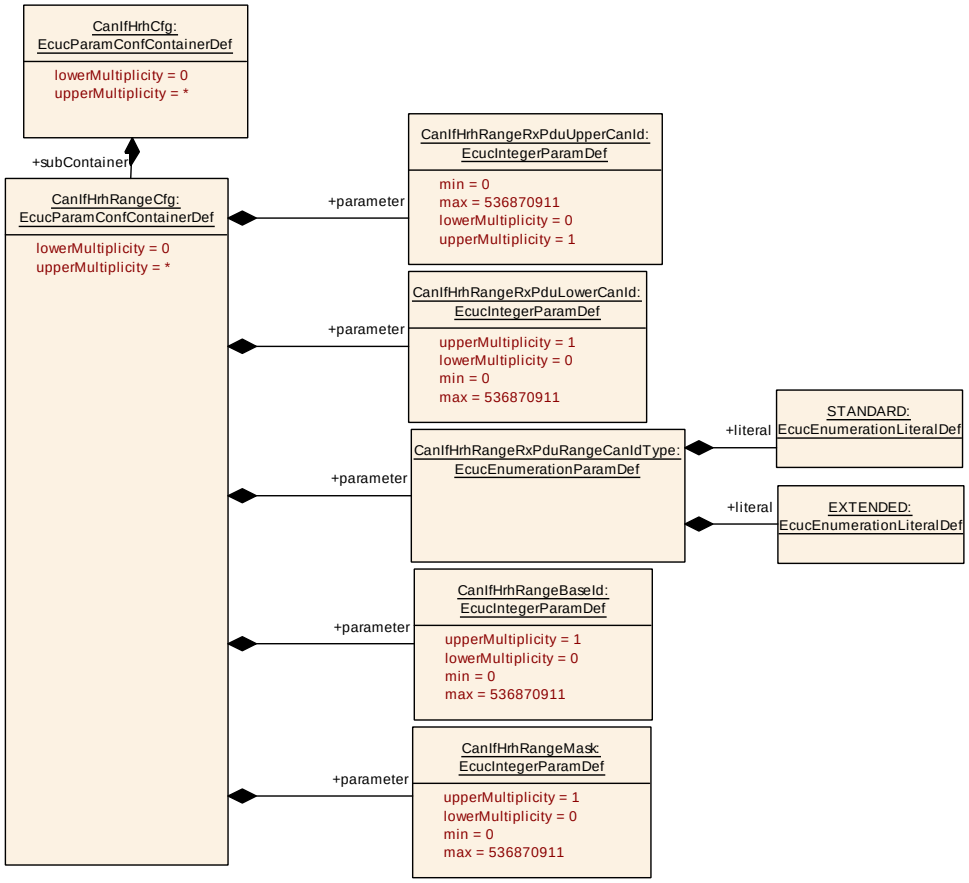


Figure 10.19: AR_EcucDef_CanIfHrhRangeCfg

10.1.19 CanIfBufferCfg

[ECUC_CanIf_00832] Definition of EcucParamConfContainerDef CanIfBufferCfg

⌈

Container Name	CanIfBufferCfg
Parent Container	CanIfInitCfg
Upstream mapping defined	–

▽

△

Description	This container contains the Txbuffer configuration. Multiple buffers with different sizes could be configured. If CanIfBufferSize (ECUC_CanIf_00834) equals 0, the CanIf Tx L-PDU only refers via this CanIfBufferCfg the corresponding CanIfHthCfg.	
Multiplicity	0..*	
Post-Build Multiplicity	Loadable	Selectable
Configuration Parameters		

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
CanIfBufferSize	1	[ECUC_CanIf_00834]
CanIfBufferHthRef	1	[ECUC_CanIf_00833]

No Included Containers

⌋

[ECUC_CanIf_00834] Definition of EcucIntegerParamDef CanIfBufferSize ⌈

Parameter Name	CanIfBufferSize	
Parent Container	CanIfBufferCfg	
Upstream mapping defined	–	
Description	This parameter defines the number of CanIf Tx L-PDUs which can be buffered in one Txbuffer. If this value equals 0, the CanIf does not perform Txbuffering for the CanIf Tx L-PDUs which are assigned to this Txbuffer. If CanIfPublicTxBuffering equals False, this parameter equals 0 for all TxBuffer. If the CanHandleType of the referred HTH equals FULL, this parameter equals 0 for this TxBuffer. Dependency: CanIfPublicTxBuffering, CanHandleType	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_CanIf_00833] Definition of EcucReferenceDef CanIfBufferHthRef ⌈

Parameter Name	CanIfBufferHthRef	
Parent Container	CanIfBufferCfg	
Upstream mapping defined	–	
Description	Reference to HTH, that defines the hardware object or the pool of hardware objects configured for transmission. All the CanIf Tx L-PDUs refer via the CanIfBufferCfg and this parameter to the HTHs if TxBuffering is enabled, or not. Each HTH shall not be assigned to more than one buffer.	
Multiplicity	1	
Type	Reference to CanIfHthCfg	
Post-Build Value	Loadable	Selectable

⌋

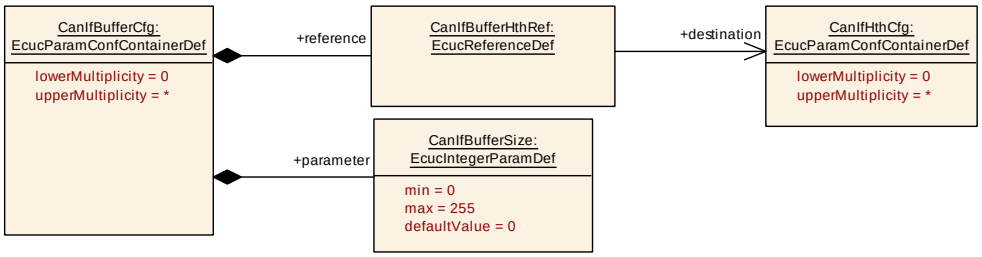


Figure 10.20: AR_EcucDef_CanIfBufferCfg

10.1.20 CanIfSecurityEventRefs

[ECUC_CanIf_00849] Definition of EcucParamConfContainerDef CanIfSecurityEventRefs

Status: DRAFT

⌈

Container Name	CanIfSecurityEventRefs
Parent Container	CanIfPublicCfg
Upstream mapping defined	–
Description	Container for the references to IdsMEvent elements representing the security events that the CanIf module shall report to the IdsM in case the coresponding security related event occurs (and if CanIfEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SEV_CAN_ERRORSTATE_BUSOFF	0..1	[ECUC_CanIf_00853]
SEV_CAN_ERRORSTATE_PASSIVE	0..1	[ECUC_CanIf_00852]
SEV_CAN_RX_ERROR_DETECTED	0..1	[ECUC_CanIf_00851]
SEV_CAN_TX_ERROR_DETECTED	0..1	[ECUC_CanIf_00850]

No Included Containers

⌋

[ECUC_CanIf_00853] Definition of EcucReferenceDef SEV_CAN_ERRORSTATE_BUSOFF

Status: DRAFT

⌈

Parameter Name	SEV_CAN_ERRORSTATE_BUSOFF
Parent Container	CanIfSecurityEventRefs
Upstream mapping defined	–
Description	The CAN controller transitioned to state busoff. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_CanIf_00852] Definition of EcucReferenceDef SEV_CAN_ERRORSTATE_PASSIVE

Status: DRAFT

⌈

Parameter Name	SEV_CAN_ERRORSTATE_PASSIVE
Parent Container	CanIfSecurityEventRefs
Upstream mapping defined	–
Description	A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_CanIf_00851] Definition of EcucReferenceDef SEV_CAN_RX_ERROR_DETECTED

Status: DRAFT

⌈

Parameter Name	SEV_CAN_RX_ERROR_DETECTED
Parent Container	CanIfSecurityEventRefs
Upstream mapping defined	–
Description	A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

[ECUC_CanIf_00850] Definition of EcucReferenceDef SEV_CAN_TX_ERROR_DETECTED

Status: DRAFT

⌈

Parameter Name	SEV_CAN_TX_ERROR_DETECTED
Parent Container	CanIfSecurityEventRefs
Upstream mapping defined	–
Description	A transmission related error was detected. Depending on the context data this could indicate suspicious CAN activity. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	Symbolic name reference to IdsMEvent
Pre-Compile only	true

⌋

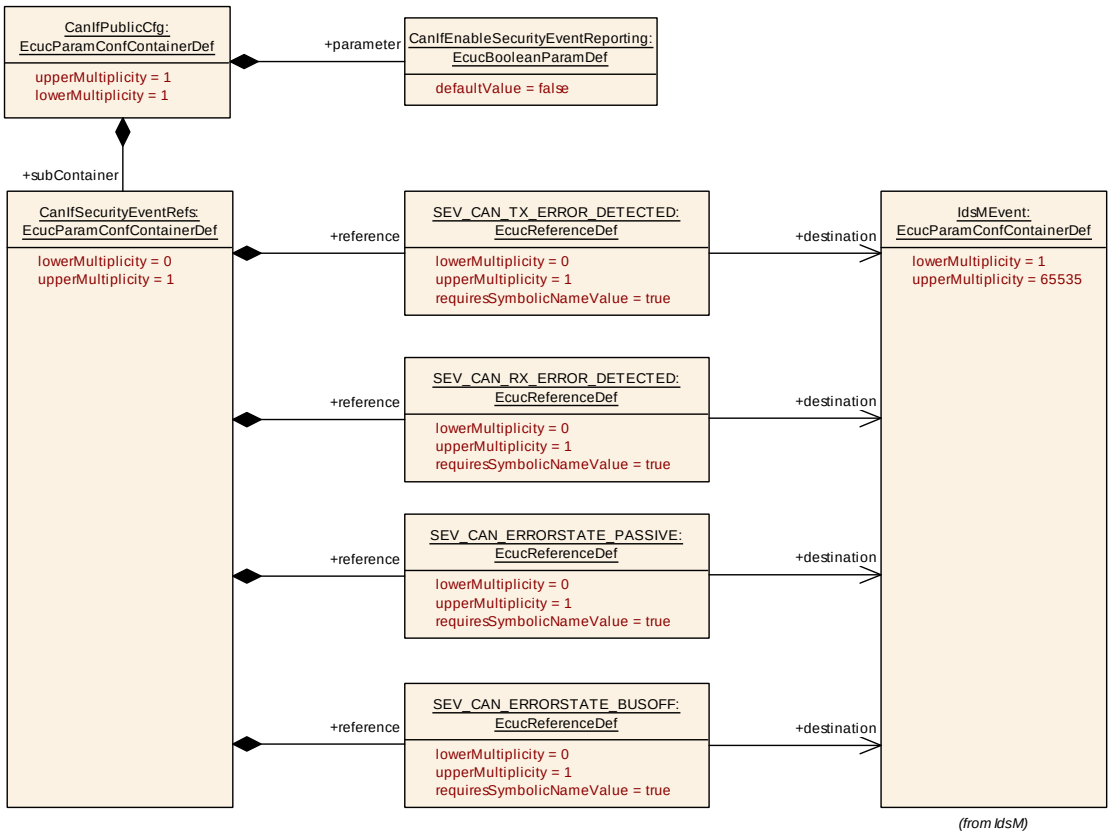


Figure 10.21: AR_EcucDef_CanIfSecurityEventRefs

10.2 Configuration constraints

[SWS_CANIF_CONSTR_00001] ⌈For each `CanIfRxPduCfg` that contains `CanIf-RxPduXLParams`, the CAN XL parameter `Priority ID` shall either be configured as

MetaData item of type PRIORITYID_16 for the global PDU referenced via CanIfRxPduRef, or as CanIfRxPduXLParams.CanIfXLPriorityId.⌋

[SWS_CANIF_CONSTR_00002] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams, the CAN XL parameter Priority ID shall either be configured as MetaData item of type PRIORITYID_16 for the global PDU referenced via CanIfTxPduRef, or as CanIfTxPduXLParams.CanIfXLPriorityId.⌋

[SWS_CANIF_CONSTR_00003] ⌈For each CanIfRxPduCfg that contains CanIfRxPduXLParams, the CAN XL parameter SDU Type shall either be configured as MetaData item of type SDUTYPE_8 for the global PDU referenced via CanIfRxPduRef, or as CanIfRxPduXLParams.CanIfXLSduType.⌋

[SWS_CANIF_CONSTR_00004] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams, the CAN XL parameter SDU Type shall either be configured as MetaData item of type SDUTYPE_8 for the global PDU referenced via CanIfTxPduRef, or as CanIfTxPduXLParams.CanIfXLSduType.⌋

[SWS_CANIF_CONSTR_00005] ⌈For each CanIfRxPduCfg that contains CanIfRxPduXLParams, the CAN XL parameter Virtual CAN Network ID shall either be configured as MetaData item of type VLAN_16 for the global PDU referenced via CanIfRxPduRef, or as CanIfRxPduXLParams.CanIfXLVcid.⌋

[SWS_CANIF_CONSTR_00006] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams, the CAN XL parameter Virtual CAN Network ID shall either be configured as MetaData item of type VLAN_16 for the global PDU referenced via CanIfTxPduRef, or as CanIfTxPduXLParams.CanIfXLVcid.⌋

[SWS_CANIF_CONSTR_00007] ⌈For each CanIfRxPduCfg that contains CanIfRxPduXLParams where the CanIfRxPduXLParams.CanIfXLSduType is configured but not set to 03h, the CAN XL parameter Acceptance Field shall either be configured as MetaData item of type ACCEPTANCEFIELD_32 for the global PDU referenced via CanIfRxPduRef, or as CanIfRxPduXLParams.CanIfXLAcceptanceField.⌋

[SWS_CANIF_CONSTR_00008] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams where the CanIfTxPduXLParams.CanIfXLSduType is configured but not set to 03h, the CAN XL parameter Acceptance Field shall either be configured as MetaData item of type ACCEPTANCEFIELD_32 for the global PDU referenced via CanIfTxPduRef, or as CanIfTxPduXLParams.CanIfXLAcceptanceField.⌋

[SWS_CANIF_CONSTR_00009] ⌈For each CanIfRxPduCfg that contains CanIfRxPduXLParams where the CanIfRxPduXLParams.CanIfXLSduType is configured as 03h, the CAN XL parameter Acceptance Field shall neither be configured as MetaData item of type ACCEPTANCEFIELD_32 for the global PDU referenced via CanIfRxPduRef, nor as CanIfRxPduXLParams.CanIfXLAcceptanceField.⌋

[SWS_CANIF_CONSTR_00010] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams where the CanIfTxPduXLParams.CanIfXLSduType is configured

as 03h, the CAN XL parameter Acceptance Field shall neither be configured as MetaData item of type ACCEPTANCEFIELD_32 for the global PDU referenced via CanIfTxPduRef, nor as CanIfTxPduXLParams.CanIfXLAcceptanceField.⌋

[SWS_CANIF_CONSTR_00011] ⌈For each CanIfRxPduCfg that contains CanIfRxPduXLParams where the CanIfRxPduXLParams.CanIfXLSduType is configured but not set to 03h, the parameters CanIfRxPduCanId and CanIfRxPduCanIdMask shall not be configured, and no MetaData item of type CAN_ID_32 shall be configured for the global PDU referenced via CanIfRxPduRef.⌋

[SWS_CANIF_CONSTR_00012] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams where the CanIfTxPduXLParams.CanIfXLSduType is configured but not set to 03h, the parameters CanIfTxPduCanId and CanIfTxPduCanIdMask shall not be configured, and no MetaData item of type CAN_ID_32 shall be configured for the global PDU referenced via CanIfTxPduRef.⌋

[SWS_CANIF_CONSTR_00013] ⌈For each CanIfRxPduCfg that contains CanIfRxPduXLParams where the parameters CanIfRxPduCanId and/or CanIfRxPduCanIdMask are configured or a MetaData item of type CAN_ID_32 is configured for the global PDU referenced via CanIfRxPduRef, the CanIfTxPduXLParams.CanIfXLSduType shall be configured as 03h.⌋

[SWS_CANIF_CONSTR_00014] ⌈For each CanIfTxPduCfg that contains CanIfTxPduXLParams where the parameters CanIfTxPduCanId and/or CanIfTxPduCanIdMask are configured or a MetaData item of type CAN_ID_32 is configured for the global PDU referenced via CanIfTxPduRef, the CanIfTxPduXLParams.CanIfXLSduType shall be configured as 03h.⌋

[SWS_CANIF_CONSTR_00015] Same configuration of PDUs that belong to the same CanIfCtrlCfg for PduBufferAlignment ⌈All CanIfTxPduCfg that belong to same CanIfCtrlCfg shall have the same value for PduBufferAlignment configured.⌋

The PDU buffer alignment of PDUs need to be harmonized with the buffer alignment of the hardware buffers of the corresponding CAN XL controller.

[SWS_CANIF_CONSTR_00016] Memory address alignment of the used hardware ⌈If a CanIfCtrlCfg references a CanXLController via CanIfCtrlCanCtrlRef, then all PDUs which are associated with this CanIfCtrlCfg shall have the same value for PduBufferAlignment configured as the referenced CanXLController (as CanXLCtrlRequiredAlignment).⌋

[SWS_CANIF_CONSTR_00017] Pdu.KeepLocalPduBuffer applicable to CAN XL PDUs only ⌈Pdu.KeepLocalPduBuffer shall only be configured for CAN XL PDUs.⌋

A Not applicable requirements

[SWS_CANIF_NA_00999]

Upstream requirements: SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00170, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00417

⌈These requirements are not applicable to this specification.⌋

B Change History

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

B.1 Change History of this document according to AUTOSAR Release R23-11

B.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_CANIF_00958]	
[SWS_CANIF_00959]	
[SWS_CANIF_00960]	
[SWS_CANIF_00961]	
[SWS_CANIF_00962]	
[SWS_CANIF_00963]	
[SWS_CANIF_00964]	
[SWS_CANIF_00965]	
[SWS_CANIF_00966]	
[SWS_CANIF_91016]	Definition of callback function CanIf_ConfirmCtrlPnAvailability
[SWS_CANIF_91017]	Definition of configurable interface <User_ConfirmCtrlPnAvailability>

Table B.1: Added Specification Items in R23-11

B.1.2 Changed Specification Items in R23-11

Number	Heading
[SWS_CANIF_00011]	Definition of configurable interface <User_TxConfirmation>
[SWS_CANIF_00012]	Definition of configurable interface <User_RxIndication>
[SWS_CANIF_00014]	Definition of configurable interface <User_ControllerBusOff>
[SWS_CANIF_00532]	Definition of configurable interface <User_ValidateWakeupEvent>
[SWS_CANIF_00687]	Definition of configurable interface <User_ControllerModeIndication>
[SWS_CANIF_00693]	Definition of configurable interface <User_TrcvModeIndication>
[SWS_CANIF_00747]	
[SWS_CANIF_00748]	
[SWS_CANIF_00750]	

▽

△

Number	Heading
[SWS_CANIF_00754]	
[SWS_CANIF_00771]	
[SWS_CANIF_00788]	Definition of configurable interface <User_ClearTrcvWufFlagIndication>
[SWS_CANIF_00794]	
[SWS_CANIF_00795]	
[SWS_CANIF_00796]	
[SWS_CANIF_00800]	
[SWS_CANIF_00801]	
[SWS_CANIF_00802]	
[SWS_CANIF_00808]	
[SWS_CANIF_00812]	
[SWS_CANIF_00813]	
[SWS_CANIF_00814]	Definition of configurable interface <User_CheckTrcvWakeFlagIndication>
[SWS_CANIF_00821]	Definition of configurable interface <User_ConfirmPnAvailability>
[SWS_CANIF_00823]	
[SWS_CANIF_00824]	
[SWS_CANIF_00825]	
[SWS_CANIF_00915]	
[SWS_CANIF_00916]	
[SWS_CANIF_00917]	
[SWS_CANIF_00918]	
[SWS_CANIF_NA_00999]	

Table B.2: Changed Specification Items in R23-11

B.1.3 Deleted Specification Items in R23-11

none

B.1.4 Added Constraints in R23-11

none

B.1.5 Changed Constraints in R23-11

none

B.1.6 Deleted Constraints in R23-11

none

B.2 Change History of this document according to AUTOSAR Release R24-11

B.2.1 Added Specification Items in R24-11

Number	Heading
[SWS_CANIF_00967]	Support of meta data type `TIMETUPLE_TYPE_PTR` at reception indication
[SWS_CANIF_92000]	Security event context data definition: SEV_CAN_TX_ERROR_DETECTED
[SWS_CANIF_92001]	Security event context data definition: SEV_CAN_RX_ERROR_DETECTED
[SWS_CANIF_92002]	Security event context data definition: SEV_CAN_ERRORSTATE_PASSIVE
[SWS_CANIF_92003]	Security event context data definition: SEV_CAN_ERRORSTATE_BUSOFF

Table B.3: Added Specification Items in R24-11

B.2.2 Changed Specification Items in R24-11

Number	Heading
[ECUC_CanIf_00247]	Definition of EcucParamConfContainerDef CanIfInitCfg
[ECUC_CanIf_00248]	Definition of EcucParamConfContainerDef CanIfTxPduCfg
[ECUC_CanIf_00249]	Definition of EcucParamConfContainerDef CanIfRxPduCfg
[ECUC_CanIf_00840]	Definition of EcucBooleanParamDef CanIfTxPduTriggerTransmit
[SWS_CANIF_00006]	Definition of callback function CanIf_RxIndication
[SWS_CANIF_00007]	Definition of callback function CanIf_TxConfirmation
[SWS_CANIF_00056]	
[SWS_CANIF_00075]	
[SWS_CANIF_00142]	Definition of imported datatypes of module CanIf
[SWS_CANIF_00218]	Definition of callback function CanIf_ControllerBusOff
[SWS_CANIF_00294]	Definition of optional interfaces requested by module CanIf
[SWS_CANIF_00383]	
[SWS_CANIF_00412]	
[SWS_CANIF_00489]	
[SWS_CANIF_00665]	
[SWS_CANIF_00666]	
[SWS_CANIF_00699]	Definition of callback function CanIf_ControllerModeIndication
[SWS_CANIF_00739]	
[SWS_CANIF_00762]	Definition of callback function CanIf_ClearTrcvWufFlagIndication
[SWS_CANIF_00763]	Definition of callback function CanIf_CheckTrcvWakeFlagIndication
[SWS_CANIF_00764]	Definition of callback function CanIf_TrcvModeIndication
[SWS_CANIF_00815]	Definition of callback function CanIf_ConfirmPnAvailability

▽

△

Number	Heading
[SWS_CANIF_00829]	
[SWS_CANIF_00830]	
[SWS_CANIF_00885]	
[SWS_CANIF_00913]	
[SWS_CANIF_00915]	
[SWS_CANIF_00916]	
[SWS_CANIF_00917]	
[SWS_CANIF_00918]	
[SWS_CANIF_00941]	
[SWS_CANIF_00948]	
[SWS_CANIF_00953]	
[SWS_CANIF_91008]	Definition of API function CanIf_ControllerErrorStatePassive
[SWS_CANIF_91009]	Definition of API function CanIf_ErrorNotification
[SWS_CANIF_91010]	Security events for CanIf
[SWS_CANIF_91013]	Definition of API function CanIf_GetIngressTimeStamp
[SWS_CANIF_91015]	Definition of API function CanIf_XLRxIndication
[SWS_CANIF_91016]	Definition of callback function CanIf_ConfirmCtrlPnAvailability

Table B.4: Changed Specification Items in R24-11

B.2.3 Deleted Specification Items in R24-11

Number	Heading
[ECUC_CanIf_00527]	Definition of EcucEnumerationParamDef CanIfTxPduUserTxConfirmationUL
[ECUC_CanIf_00528]	Definition of EcucFunctionNameDef CanIfTxPduUserTxConfirmationName
[ECUC_CanIf_00529]	Definition of EcucEnumerationParamDef CanIfRxPduUserRxIndicationUL
[ECUC_CanIf_00530]	Definition of EcucFunctionNameDef CanIfRxPduUserRxIndicationName
[ECUC_CanIf_00842]	Definition of EcucFunctionNameDef CanIfTxPduUserTriggerTransmitName
[SWS_CANIF_00011]	Definition of configurable interface <User_TxConfirmation>
[SWS_CANIF_00012]	Definition of configurable interface <User_RxIndication>
[SWS_CANIF_00135]	
[SWS_CANIF_00414]	
[SWS_CANIF_00423]	
[SWS_CANIF_00438]	
[SWS_CANIF_00439]	
[SWS_CANIF_00441]	
[SWS_CANIF_00442]	

▽

△

Number	Heading
[SWS_CANIF_00445]	
[SWS_CANIF_00448]	
[SWS_CANIF_00542]	
[SWS_CANIF_00543]	
[SWS_CANIF_00544]	
[SWS_CANIF_00550]	
[SWS_CANIF_00551]	
[SWS_CANIF_00552]	
[SWS_CANIF_00554]	
[SWS_CANIF_00555]	
[SWS_CANIF_00556]	
[SWS_CANIF_00557]	
[SWS_CANIF_00858]	
[SWS_CANIF_00859]	
[SWS_CANIF_00879]	
[SWS_CANIF_00880]	
[SWS_CANIF_00886]	Definition of configurable interface <User_TriggerTransmit>
[SWS_CANIF_00888]	
[SWS_CANIF_00889]	
[SWS_CANIF_00890]	
[SWS_CANIF_00891]	

Table B.5: Deleted Specification Items in R24-11

B.2.4 Added Constraints in R24-11

none

B.2.5 Changed Constraints in R24-11

none

B.2.6 Deleted Constraints in R24-11

none

B.3 Change History of this document according to AUTOSAR Release R25-11

B.3.1 Added Specification Items in R25-11

Number	Heading
[SWS_CANIF_00968]	
[SWS_CANIF_00969]	
[SWS_CANIF_91018]	Definition of API function CanIf_XLReportErrorState

Table B.6: Added Specification Items in R25-11

B.3.2 Changed Specification Items in R25-11

Number	Heading
[ECUC_CanIf_00244]	Definition of EcucModuleDef CanIf
[ECUC_CanIf_00245]	Definition of EcucParamConfContainerDef CanIfPrivateCfg
[ECUC_CanIf_00246]	Definition of EcucParamConfContainerDef CanIfPublicCfg
[ECUC_CanIf_00247]	Definition of EcucParamConfContainerDef CanIfInitCfg
[ECUC_CanIf_00248]	Definition of EcucParamConfContainerDef CanIfTxPduCfg
[ECUC_CanIf_00249]	Definition of EcucParamConfContainerDef CanIfRxPduCfg
[ECUC_CanIf_00250]	Definition of EcucParamConfContainerDef CanIfDispatchCfg
[ECUC_CanIf_00253]	Definition of EcucParamConfContainerDef CanIfCtrlDrvCfg
[ECUC_CanIf_00257]	Definition of EcucParamConfContainerDef CanIfInitHohCfg
[ECUC_CanIf_00258]	Definition of EcucParamConfContainerDef CanIfHthCfg
[ECUC_CanIf_00259]	Definition of EcucParamConfContainerDef CanIfHrhCfg
[ECUC_CanIf_00273]	Definition of EcucParamConfContainerDef CanIfTrcvDrvCfg
[ECUC_CanIf_00525]	Definition of EcucFunctionNameDef CanIfDispatchUserCtrlBusOffName
[ECUC_CanIf_00531]	Definition of EcucFunctionNameDef CanIfDispatchUserValidateWakeup EventName
[ECUC_CanIf_00546]	Definition of EcucParamConfContainerDef CanIfCtrlCfg
[ECUC_CanIf_00547]	Definition of EcucEnumerationParamDef CanIfDispatchUserCtrlBusOffUL
[ECUC_CanIf_00549]	Definition of EcucEnumerationParamDef CanIfDispatchUserValidateWakeup EventUL
[ECUC_CanIf_00587]	Definition of EcucParamConfContainerDef CanIfTrcvCfg
[ECUC_CanIf_00589]	Definition of EcucBooleanParamDef CanIfTxPduReadNotifyStatus
[ECUC_CanIf_00591]	Definition of EcucIntegerParamDef CanIfTxPduId
[ECUC_CanIf_00592]	Definition of EcucIntegerParamDef CanIfTxPduCanId
[ECUC_CanIf_00593]	Definition of EcucEnumerationParamDef CanIfTxPduType
[ECUC_CanIf_00595]	Definition of EcucBooleanParamDef CanIfRxPduReadNotifyStatus

▽

△

Number	Heading
[ECUC_CanIf_00597]	Definition of EcucIntegerParamDef CanIfRxPduId
[ECUC_CanIf_00598]	Definition of EcucIntegerParamDef CanIfRxPduCanId
[ECUC_CanIf_00599]	Definition of EcucIntegerParamDef CanIfRxPduDataLength
[ECUC_CanIf_00600]	Definition of EcucBooleanParamDef CanIfRxPduReadData
[ECUC_CanIf_00601]	Definition of EcucReferenceDef CanIfRxPduRef
[ECUC_CanIf_00602]	Definition of EcucReferenceDef CanIfRxPduHrhIdRef
[ECUC_CanIf_00603]	Definition of EcucReferenceDef CanIfTxPduRef
[ECUC_CanIf_00605]	Definition of EcucReferenceDef CanIfTrcvCanTrcvRef
[ECUC_CanIf_00606]	Definition of EcucBooleanParamDef CanIfTrcvWakeupSupport
[ECUC_CanIf_00607]	Definition of EcucBooleanParamDef CanIfPublicReadRxPduDataApi
[ECUC_CanIf_00608]	Definition of EcucBooleanParamDef CanIfPublicReadRxPduNotifyStatusApi
[ECUC_CanIf_00609]	Definition of EcucBooleanParamDef CanIfPublicReadTxPduNotifyStatusApi
[ECUC_CanIf_00610]	Definition of EcucBooleanParamDef CanIfPublicSetDynamicTxIdApi
[ECUC_CanIf_00611]	Definition of EcucBooleanParamDef CanIfPublicWakeupCheckValidSupport
[ECUC_CanIf_00612]	Definition of EcucBooleanParamDef CanIfPublicMultipleDrvSupport
[ECUC_CanIf_00613]	Definition of EcucBooleanParamDef CanIfVersionInfoApi
[ECUC_CanIf_00614]	Definition of EcucBooleanParamDef CanIfDevErrorDetect
[ECUC_CanIf_00617]	Definition of EcucBooleanParamDef CanIfPrivateDataLengthCheck
[ECUC_CanIf_00618]	Definition of EcucBooleanParamDef CanIfPublicTxBuffering
[ECUC_CanIf_00619]	Definition of EcucEnumerationParamDef CanIfPrivateSoftwareFilterType
[ECUC_CanIf_00623]	Definition of EcucStringParamDef CanIfInitCfgSet
[ECUC_CanIf_00625]	Definition of EcucReferenceDef CanIfHthCanCtrlIdRef
[ECUC_CanIf_00627]	Definition of EcucChoiceReferenceDef CanIfHthIdSymRef
[ECUC_CanIf_00628]	Definition of EcucParamConfContainerDef CanIfHrhRangeCfg
[ECUC_CanIf_00629]	Definition of EcucIntegerParamDef CanIfHrhRangeRxPduLowerCanId
[ECUC_CanIf_00630]	Definition of EcucIntegerParamDef CanIfHrhRangeRxPduUpperCanId
[ECUC_CanIf_00631]	Definition of EcucReferenceDef CanIfHrhCanCtrlIdRef
[ECUC_CanIf_00632]	Definition of EcucBooleanParamDef CanIfHrhSoftwareFilter
[ECUC_CanIf_00634]	Definition of EcucChoiceReferenceDef CanIfHrhIdSymRef
[ECUC_CanIf_00636]	Definition of EcucReferenceDef CanIfCtrlCanCtrlRef
[ECUC_CanIf_00637]	Definition of EcucBooleanParamDef CanIfCtrlWakeupSupport
[ECUC_CanIf_00638]	Definition of EcucReferenceDef CanIfCtrlDrvNameRef
[ECUC_CanIf_00642]	Definition of EcucReferenceDef CanIfCtrlDrvInitHohConfigRef
[ECUC_CanIf_00644]	Definition of EcucEnumerationParamDef CanIfHrhRangeRxPduRangeCanIdType
[ECUC_CanIf_00647]	Definition of EcucIntegerParamDef CanIfCtrlId
[ECUC_CanIf_00654]	Definition of EcucIntegerParamDef CanIfTrcvId
[ECUC_CanIf_00671]	Definition of EcucStringParamDef CanIfPublicCddHeaderFile

▽

△

Number	Heading
[ECUC_CanIf_00683]	Definition of EcucFunctionNameDef CanIfDispatchUserCtrlModeIndication Name
[ECUC_CanIf_00684]	Definition of EcucEnumerationParamDef CanIfDispatchUserCtrlMode IndicationUL
[ECUC_CanIf_00685]	Definition of EcucFunctionNameDef CanIfDispatchUserTrcvModeIndication Name
[ECUC_CanIf_00686]	Definition of EcucEnumerationParamDef CanIfDispatchUserTrcvMode IndicationUL
[ECUC_CanIf_00733]	Definition of EcucBooleanParamDef CanIfPublicTxConfirmPollingSupport
[ECUC_CanIf_00741]	Definition of EcucBooleanParamDef CanIfPublicWakeupCheckValidByNM
[ECUC_CanIf_00742]	Definition of EcucEnumerationParamDef CanIfPublicHandleTypeEnum
[ECUC_CanIf_00743]	Definition of EcucParamConfContainerDef CanIfRxPduCanIdRange
[ECUC_CanIf_00744]	Definition of EcucIntegerParamDef CanIfRxPduCanIdRangeUpperCanId
[ECUC_CanIf_00745]	Definition of EcucIntegerParamDef CanIfRxPduCanIdRangeLowerCanId
[ECUC_CanIf_00772]	Definition of EcucBooleanParamDef CanIfPublicPnSupport
[ECUC_CanIf_00773]	Definition of EcucBooleanParamDef CanIfTxPduPnFilterPdu
[ECUC_CanIf_00789]	Definition of EcucFunctionNameDef CanIfDispatchUserClearTrcvWufFlag IndicationName
[ECUC_CanIf_00790]	Definition of EcucEnumerationParamDef CanIfDispatchUserClearTrcvWuf FlagIndicationUL
[ECUC_CanIf_00791]	Definition of EcucFunctionNameDef CanIfDispatchUserCheckTrcvWakeFlag IndicationName
[ECUC_CanIf_00792]	Definition of EcucEnumerationParamDef CanIfDispatchUserCheckTrcvWake FlagIndicationUL
[ECUC_CanIf_00819]	Definition of EcucFunctionNameDef CanIfDispatchUserConfirmPn AvailabilityName
[ECUC_CanIf_00820]	Definition of EcucEnumerationParamDef CanIfDispatchUserConfirmPn AvailabilityUL
[ECUC_CanIf_00822]	Definition of EcucIntegerParamDef CanIfRxPduCanIdMask
[ECUC_CanIf_00823]	Definition of EcucIntegerParamDef CanIfTxPduCanIdMask
[ECUC_CanIf_00824]	Definition of EcucBooleanParamDef CanIfMetaDataSupport
[ECUC_CanIf_00825]	Definition of EcucIntegerParamDef CanIfHrhRangeBaseId
[ECUC_CanIf_00826]	Definition of EcucIntegerParamDef CanIfHrhRangeMask
[ECUC_CanIf_00827]	Definition of EcucBooleanParamDef CanIfFixedBuffer
[ECUC_CanIf_00828]	Definition of EcucIntegerParamDef CanIfMaxBufferSize
[ECUC_CanIf_00829]	Definition of EcucIntegerParamDef CanIfMaxTxPduCfg
[ECUC_CanIf_00830]	Definition of EcucIntegerParamDef CanIfMaxRxPduCfg
[ECUC_CanIf_00831]	Definition of EcucReferenceDef CanIfTxPduBufferRef
[ECUC_CanIf_00832]	Definition of EcucParamConfContainerDef CanIfBufferCfg
[ECUC_CanIf_00833]	Definition of EcucReferenceDef CanIfBufferHthRef
[ECUC_CanIf_00834]	Definition of EcucIntegerParamDef CanIfBufferSize

▽

△

Number	Heading
[ECUC_CanIf_00837]	Definition of EcucBooleanParamDef CanIfTxOfflineActiveSupport
[ECUC_CanIf_00838]	Definition of EcucBooleanParamDef CanIfSetBaudrateApi
[ECUC_CanIf_00840]	Definition of EcucBooleanParamDef CanIfTxPduTriggerTransmit
[ECUC_CanIf_00843]	Definition of EcucBooleanParamDef CanIfWakeupSupport
[ECUC_CanIf_00844]	Definition of EcucBooleanParamDef CanIfTriggerTransmitSupport
[ECUC_CanIf_00845]	Definition of EcucBooleanParamDef CanIfTxPduTruncation
[ECUC_CanIf_00846]	Definition of EcucBooleanParamDef CanIfRxPduDataLengthCheck
[ECUC_CanIf_00847]	Definition of EcucBooleanParamDef CanIfBusMirroringSupport
[ECUC_CanIf_00848]	Definition of EcucBooleanParamDef CanIfEnableSecurityEventReporting
[ECUC_CanIf_00849]	Definition of EcucParamConfContainerDef CanIfSecurityEventRefs
[ECUC_CanIf_00850]	Definition of EcucReferenceDef SEV_CAN_TX_ERROR_DETECTED
[ECUC_CanIf_00851]	Definition of EcucReferenceDef SEV_CAN_RX_ERROR_DETECTED
[ECUC_CanIf_00852]	Definition of EcucReferenceDef SEV_CAN_ERRORSTATE_PASSIVE
[ECUC_CanIf_00853]	Definition of EcucReferenceDef SEV_CAN_ERRORSTATE_BUSOFF
[ECUC_CanIf_00854]	Definition of EcucBooleanParamDef CanIfGlobalTimeSupport
[ECUC_CanIf_00855]	Definition of EcucParamConfContainerDef CanIfTxPduXLParams
[ECUC_CanIf_00856]	Definition of EcucIntegerParamDef CanIfXLPriorityId
[ECUC_CanIf_00857]	Definition of EcucIntegerParamDef CanIfXLVcid
[ECUC_CanIf_00858]	Definition of EcucIntegerParamDef CanIfXLSduType
[ECUC_CanIf_00859]	Definition of EcucIntegerParamDef CanIfXLAcceptanceField
[ECUC_CanIf_00860]	Definition of EcucParamConfContainerDef CanIfRxPduXLParams
[ECUC_CanIf_00865]	Definition of EcucBooleanParamDef CanIfPublicTrcvPnEnable
[ECUC_CanIf_00866]	Definition of EcucBooleanParamDef CanIfPublicCtrlPnEnable
[ECUC_CanIf_00867]	Definition of EcucFunctionNameDef CanIfDispatchUserConfirmCtrlPn AvailabilityName
[ECUC_CanIf_00868]	Definition of EcucEnumerationParamDef CanIfDispatchUserConfirmCtrlPn AvailabilityUL
[SWS_CANIF_00026]	
[SWS_CANIF_00033]	
[SWS_CANIF_00070]	
[SWS_CANIF_00072]	
[SWS_CANIF_00073]	
[SWS_CANIF_00075]	
[SWS_CANIF_00115]	
[SWS_CANIF_00118]	
[SWS_CANIF_00137]	Definition of datatype CanIf_PduModeType
[SWS_CANIF_00142]	Definition of imported datatypes of module CanIf
[SWS_CANIF_00144]	Definition of datatype CanIf_ConfigType
[SWS_CANIF_00183]	

▽

△

Number	Heading
[SWS_CANIF_00188]	
[SWS_CANIF_00201]	Definition of datatype CanIf_NotifStatusType
[SWS_CANIF_00294]	Definition of optional interfaces requested by module CanIf
[SWS_CANIF_00378]	
[SWS_CANIF_00386]	
[SWS_CANIF_00395]	
[SWS_CANIF_00467]	
[SWS_CANIF_00489]	
[SWS_CANIF_00645]	
[SWS_CANIF_00646]	
[SWS_CANIF_00662]	
[SWS_CANIF_00664]	
[SWS_CANIF_00667]	
[SWS_CANIF_00668]	
[SWS_CANIF_00748]	
[SWS_CANIF_00750]	
[SWS_CANIF_00844]	
[SWS_CANIF_00847]	
[SWS_CANIF_00849]	
[SWS_CANIF_00854]	
[SWS_CANIF_00857]	
[SWS_CANIF_00877]	
[SWS_CANIF_00942]	

Table B.7: Changed Specification Items in R25-11

B.3.3 Deleted Specification Items in R25-11

Number	Heading
[ECUC_CanIf_00590]	Definition of EcucEnumerationParamDef CanIfTxPduCanIdType
[ECUC_CanIf_00596]	Definition of EcucEnumerationParamDef CanIfRxPduCanIdType
[ECUC_CanIf_00675]	Definition of EcucBooleanParamDef CanIfSupportTTCAN
[SWS_CANIF_00243]	
[SWS_CANIF_00895]	

Table B.8: Deleted Specification Items in R25-11

B.3.4 Added Constraints in R25-11

none

B.3.5 Changed Constraints in R25-11

Number	Heading
[SWS_CANIF_CONSTR_00011]	
[SWS_CANIF_CONSTR_00012]	

Table B.9: Changed Constraints in R25-11

B.3.6 Deleted Constraints in R25-11

none

B.4 Change History of this document according to AUTOSAR Release R26-11

B.4.1 Added Specification Items in R26-11

Number	Heading
[SWS_CANIF_00970]	Forwarding `CANsec`-related calls to the `CanDrv`
[SWS_CANIF_00971]	Forwarding of `CANsec` related callback APIs calls towards `MKA`
[SWS_CANIF_00972]	
[SWS_CANIF_00973]	Forwarding of CAN XL reception
[SWS_CANIF_00974]	Release hardware reception buffer
[SWS_CanIf_91019]	Definition of API function CanIf_CanSecUpdateSecY
[SWS_CanIf_91020]	Definition of API function CanIf_CanSecAddRxSc
[SWS_CanIf_91021]	Definition of API function CanIf_CanSecDeleteRxSc
[SWS_CanIf_91022]	Definition of API function CanIf_CanSecAddTxSa
[SWS_CanIf_91023]	Definition of API function CanIf_CanSecUpdateTxSa
[SWS_CanIf_91024]	Definition of API function CanIf_CanSecDeleteTxSa
[SWS_CanIf_91025]	Definition of API function CanIf_CanSecAddRxSa
[SWS_CanIf_91026]	Definition of API function CanIf_CanSecUpdateRxSa
[SWS_CanIf_91027]	Definition of API function CanIf_CanSecDeleteRxSa
[SWS_CanIf_91028]	Definition of API function CanIf_CanSecGetTxSaNextPn
[SWS_CanIf_91029]	Definition of API function CanIf_CanSecGetCanSecSecYStatistics
[SWS_CanIf_91030]	Definition of API function CanIf_CanSecGetCanSecTxScStatistics
[SWS_CanIf_91031]	Definition of API function CanIf_CanSecGetCanSecRxScStatistics
[SWS_CanIf_91032]	Definition of API function CanIf_CanSecSetControlledPortEnabled
[SWS_CanIf_91033]	Definition of callback function CanIf_CanSecUpdateSecYNotification
[SWS_CanIf_91034]	Definition of callback function CanIf_CanSecAddTxSaNotification
[SWS_CanIf_91035]	Definition of callback function CanIf_CanSecAddRxSaNotification
[SWS_CanIf_91036]	Definition of callback function CanIf_CanSecGetCanSecSecYStatistics Notification
[SWS_CanIf_91037]	Definition of callback function CanIf_CanSecGetCanSecTxScStatistics Notification
[SWS_CanIf_91038]	Definition of callback function CanIf_CanSecGetCanSecRxScStatistics Notification
[SWS_CanIf_91040]	Definition of API function CanIf_ReleaseRxBuffer

Table B.10: Added Specification Items in R26-11

B.4.2 Changed Specification Items in R26-11

Number	Heading
[ECUC_CanIf_00244]	Definition of EcucModuleDef CanIf
[ECUC_CanIf_00245]	Definition of EcucParamConfContainerDef CanIfPrivateCfg
[ECUC_CanIf_00246]	Definition of EcucParamConfContainerDef CanIfPublicCfg
[ECUC_CanIf_00247]	Definition of EcucParamConfContainerDef CanIfInitCfg
[ECUC_CanIf_00248]	Definition of EcucParamConfContainerDef CanIfTxPduCfg
[ECUC_CanIf_00249]	Definition of EcucParamConfContainerDef CanIfRxPduCfg
[ECUC_CanIf_00250]	Definition of EcucParamConfContainerDef CanIfDispatchCfg
[ECUC_CanIf_00253]	Definition of EcucParamConfContainerDef CanIfCtrlDrvCfg
[ECUC_CanIf_00257]	Definition of EcucParamConfContainerDef CanIfInitHohCfg
[ECUC_CanIf_00258]	Definition of EcucParamConfContainerDef CanIfHthCfg
[ECUC_CanIf_00259]	Definition of EcucParamConfContainerDef CanIfHrhCfg
[ECUC_CanIf_00273]	Definition of EcucParamConfContainerDef CanIfTrcvDrvCfg
[ECUC_CanIf_00525]	Definition of EcucFunctionNameDef CanIfDispatchUserCtrlBusOffName
[ECUC_CanIf_00531]	Definition of EcucFunctionNameDef CanIfDispatchUserValidateWakeup EventName
[ECUC_CanIf_00546]	Definition of EcucParamConfContainerDef CanIfCtrlCfg
[ECUC_CanIf_00547]	Definition of EcucEnumerationParamDef CanIfDispatchUserCtrlBusOffUL
[ECUC_CanIf_00549]	Definition of EcucEnumerationParamDef CanIfDispatchUserValidateWakeup EventUL
[ECUC_CanIf_00587]	Definition of EcucParamConfContainerDef CanIfTrcvCfg
[ECUC_CanIf_00589]	Definition of EcucBooleanParamDef CanIfTxPduReadNotifyStatus
[ECUC_CanIf_00591]	Definition of EcucIntegerParamDef CanIfTxPduId
[ECUC_CanIf_00592]	Definition of EcucIntegerParamDef CanIfTxPduCanId
[ECUC_CanIf_00593]	Definition of EcucEnumerationParamDef CanIfTxPduType
[ECUC_CanIf_00595]	Definition of EcucBooleanParamDef CanIfRxPduReadNotifyStatus
[ECUC_CanIf_00597]	Definition of EcucIntegerParamDef CanIfRxPduId
[ECUC_CanIf_00598]	Definition of EcucIntegerParamDef CanIfRxPduCanId
[ECUC_CanIf_00599]	Definition of EcucIntegerParamDef CanIfRxPduDataLength
[ECUC_CanIf_00600]	Definition of EcucBooleanParamDef CanIfRxPduReadData
[ECUC_CanIf_00601]	Definition of EcucReferenceDef CanIfRxPduRef
[ECUC_CanIf_00602]	Definition of EcucReferenceDef CanIfRxPduHrhIdRef
[ECUC_CanIf_00603]	Definition of EcucReferenceDef CanIfTxPduRef
[ECUC_CanIf_00605]	Definition of EcucReferenceDef CanIfTrcvCanTrcvRef
[ECUC_CanIf_00606]	Definition of EcucBooleanParamDef CanIfTrcvWakeupSupport
[ECUC_CanIf_00607]	Definition of EcucBooleanParamDef CanIfPublicReadRxPduDataApi
[ECUC_CanIf_00608]	Definition of EcucBooleanParamDef CanIfPublicReadRxPduNotifyStatusApi

▽

△

Number	Heading
[ECUC_CanIf_00609]	Definition of EcucBooleanParamDef CanIfPublicReadTxPduNotifyStatusApi
[ECUC_CanIf_00610]	Definition of EcucBooleanParamDef CanIfPublicSetDynamicTxIdApi
[ECUC_CanIf_00611]	Definition of EcucBooleanParamDef CanIfPublicWakeupCheckValidSupport
[ECUC_CanIf_00612]	Definition of EcucBooleanParamDef CanIfPublicMultipleDrvSupport
[ECUC_CanIf_00614]	Definition of EcucBooleanParamDef CanIfDevErrorDetect
[ECUC_CanIf_00617]	Definition of EcucBooleanParamDef CanIfPrivateDataLengthCheck
[ECUC_CanIf_00618]	Definition of EcucBooleanParamDef CanIfPublicTxBuffering
[ECUC_CanIf_00619]	Definition of EcucEnumerationParamDef CanIfPrivateSoftwareFilterType
[ECUC_CanIf_00623]	Definition of EcucStringParamDef CanIfInitCfgSet
[ECUC_CanIf_00625]	Definition of EcucReferenceDef CanIfHthCanCtrlIdRef
[ECUC_CanIf_00627]	Definition of EcucChoiceReferenceDef CanIfHthIdSymRef
[ECUC_CanIf_00628]	Definition of EcucParamConfContainerDef CanIfHrhRangeCfg
[ECUC_CanIf_00629]	Definition of EcucIntegerParamDef CanIfHrhRangeRxPduLowerCanId
[ECUC_CanIf_00630]	Definition of EcucIntegerParamDef CanIfHrhRangeRxPduUpperCanId
[ECUC_CanIf_00631]	Definition of EcucReferenceDef CanIfHrhCanCtrlIdRef
[ECUC_CanIf_00632]	Definition of EcucBooleanParamDef CanIfHrhSoftwareFilter
[ECUC_CanIf_00634]	Definition of EcucChoiceReferenceDef CanIfHrhIdSymRef
[ECUC_CanIf_00636]	Definition of EcucReferenceDef CanIfCtrlCanCtrlRef
[ECUC_CanIf_00637]	Definition of EcucBooleanParamDef CanIfCtrlWakeupSupport
[ECUC_CanIf_00638]	Definition of EcucReferenceDef CanIfCtrlDrvNameRef
[ECUC_CanIf_00642]	Definition of EcucReferenceDef CanIfCtrlDrvInitHohConfigRef
[ECUC_CanIf_00644]	Definition of EcucEnumerationParamDef CanIfHrhRangeRxPduRangeCanId Type
[ECUC_CanIf_00647]	Definition of EcucIntegerParamDef CanIfCtrlId
[ECUC_CanIf_00654]	Definition of EcucIntegerParamDef CanIfTrcvId
[ECUC_CanIf_00671]	Definition of EcucStringParamDef CanIfPublicCddHeaderFile
[ECUC_CanIf_00683]	Definition of EcucFunctionNameDef CanIfDispatchUserCtrlModeIndication Name
[ECUC_CanIf_00684]	Definition of EcucEnumerationParamDef CanIfDispatchUserCtrlMode IndicationUL
[ECUC_CanIf_00685]	Definition of EcucFunctionNameDef CanIfDispatchUserTrcvModeIndication Name
[ECUC_CanIf_00686]	Definition of EcucEnumerationParamDef CanIfDispatchUserTrcvMode IndicationUL
[ECUC_CanIf_00733]	Definition of EcucBooleanParamDef CanIfPublicTxConfirmPollingSupport
[ECUC_CanIf_00741]	Definition of EcucBooleanParamDef CanIfPublicWakeupCheckValidByNM
[ECUC_CanIf_00742]	Definition of EcucEnumerationParamDef CanIfPublicHandleTypeEnum
[ECUC_CanIf_00743]	Definition of EcucParamConfContainerDef CanIfRxPduCanIdRange
[ECUC_CanIf_00744]	Definition of EcucIntegerParamDef CanIfRxPduCanIdRangeUpperCanId
[ECUC_CanIf_00745]	Definition of EcucIntegerParamDef CanIfRxPduCanIdRangeLowerCanId

▽

△

Number	Heading
[ECUC_CanIf_00772]	Definition of EcucBooleanParamDef CanIfPublicPnSupport
[ECUC_CanIf_00773]	Definition of EcucBooleanParamDef CanIfTxPduPnFilterPdu
[ECUC_CanIf_00789]	Definition of EcucFunctionNameDef CanIfDispatchUserClearTrcvWufFlag IndicationName
[ECUC_CanIf_00790]	Definition of EcucEnumerationParamDef CanIfDispatchUserClearTrcvWuf FlagIndicationUL
[ECUC_CanIf_00791]	Definition of EcucFunctionNameDef CanIfDispatchUserCheckTrcvWakeFlag IndicationName
[ECUC_CanIf_00792]	Definition of EcucEnumerationParamDef CanIfDispatchUserCheckTrcvWake FlagIndicationUL
[ECUC_CanIf_00819]	Definition of EcucFunctionNameDef CanIfDispatchUserConfirmPn AvailabilityName
[ECUC_CanIf_00820]	Definition of EcucEnumerationParamDef CanIfDispatchUserConfirmPn AvailabilityUL
[ECUC_CanIf_00822]	Definition of EcucIntegerParamDef CanIfRxPduCanIdMask
[ECUC_CanIf_00823]	Definition of EcucIntegerParamDef CanIfTxPduCanIdMask
[ECUC_CanIf_00824]	Definition of EcucBooleanParamDef CanIfMetaDataSupport
[ECUC_CanIf_00825]	Definition of EcucIntegerParamDef CanIfHrhRangeBaseId
[ECUC_CanIf_00826]	Definition of EcucIntegerParamDef CanIfHrhRangeMask
[ECUC_CanIf_00827]	Definition of EcucBooleanParamDef CanIfFixedBuffer
[ECUC_CanIf_00828]	Definition of EcucIntegerParamDef CanIfMaxBufferSize
[ECUC_CanIf_00829]	Definition of EcucIntegerParamDef CanIfMaxTxPduCfg
[ECUC_CanIf_00830]	Definition of EcucIntegerParamDef CanIfMaxRxPduCfg
[ECUC_CanIf_00831]	Definition of EcucReferenceDef CanIfTxPduBufferRef
[ECUC_CanIf_00832]	Definition of EcucParamConfContainerDef CanIfBufferCfg
[ECUC_CanIf_00833]	Definition of EcucReferenceDef CanIfBufferHthRef
[ECUC_CanIf_00834]	Definition of EcucIntegerParamDef CanIfBufferSize
[ECUC_CanIf_00837]	Definition of EcucBooleanParamDef CanIfTxOfflineActiveSupport
[ECUC_CanIf_00838]	Definition of EcucBooleanParamDef CanIfSetBaudrateApi
[ECUC_CanIf_00840]	Definition of EcucBooleanParamDef CanIfTxPduTriggerTransmit
[ECUC_CanIf_00843]	Definition of EcucBooleanParamDef CanIfWakeupSupport
[ECUC_CanIf_00844]	Definition of EcucBooleanParamDef CanIfTriggerTransmitSupport
[ECUC_CanIf_00845]	Definition of EcucBooleanParamDef CanIfTxPduTruncation
[ECUC_CanIf_00846]	Definition of EcucBooleanParamDef CanIfRxPduDataLengthCheck
[ECUC_CanIf_00847]	Definition of EcucBooleanParamDef CanIfBusMirroringSupport
[ECUC_CanIf_00848]	Definition of EcucBooleanParamDef CanIfEnableSecurityEventReporting
[ECUC_CanIf_00849]	Definition of EcucParamConfContainerDef CanIfSecurityEventRefs
[ECUC_CanIf_00850]	Definition of EcucReferenceDef SEV_CAN_TX_ERROR_DETECTED
[ECUC_CanIf_00851]	Definition of EcucReferenceDef SEV_CAN_RX_ERROR_DETECTED
[ECUC_CanIf_00852]	Definition of EcucReferenceDef SEV_CAN_ERRORSTATE_PASSIVE

▽

△

Number	Heading
[ECUC_CanIf_00853]	Definition of EcucReferenceDef SEV_CAN_ERRORSTATE_BUSOFF
[ECUC_CanIf_00854]	Definition of EcucBooleanParamDef CanIfGlobalTimeSupport
[ECUC_CanIf_00855]	Definition of EcucParamConfContainerDef CanIfTxPduXLParams
[ECUC_CanIf_00856]	Definition of EcucIntegerParamDef CanIfXLPriorityId
[ECUC_CanIf_00857]	Definition of EcucIntegerParamDef CanIfXLVcid
[ECUC_CanIf_00858]	Definition of EcucIntegerParamDef CanIfXLSduType
[ECUC_CanIf_00859]	Definition of EcucIntegerParamDef CanIfXLAcceptanceField
[ECUC_CanIf_00860]	Definition of EcucParamConfContainerDef CanIfRxPduXLParams
[ECUC_CanIf_00865]	Definition of EcucBooleanParamDef CanIfPublicTrcvPnEnable
[ECUC_CanIf_00866]	Definition of EcucBooleanParamDef CanIfPublicCtrlPnEnable
[ECUC_CanIf_00867]	Definition of EcucFunctionNameDef CanIfDispatchUserConfirmCtrlPn AvailabilityName
[ECUC_CanIf_00868]	Definition of EcucEnumerationParamDef CanIfDispatchUserConfirmCtrlPn AvailabilityUL
[SWS_CANIF_00001]	Definition of API function CanIf_Init
[SWS_CANIF_00003]	Definition of API function CanIf_SetControllerMode
[SWS_CANIF_00005]	Definition of API function CanIf_Transmit
[SWS_CANIF_00007]	Definition of callback function CanIf_TxConfirmation
[SWS_CANIF_00014]	Definition of configurable interface <User_ControllerBusOff>
[SWS_CANIF_00023]	
[SWS_CANIF_00026]	
[SWS_CANIF_00030]	
[SWS_CANIF_00033]	
[SWS_CANIF_00056]	Forwarding of CAN 2.0 and CAN FD reception
[SWS_CANIF_00063]	
[SWS_CANIF_00068]	
[SWS_CANIF_00075]	
[SWS_CANIF_00142]	Definition of imported datatypes of module CanIf
[SWS_CANIF_00162]	
[SWS_CANIF_00179]	
[SWS_CANIF_00194]	Definition of API function CanIf_ReadRxPduData
[SWS_CANIF_00202]	Definition of API function CanIf_ReadTxNotifStatus
[SWS_CANIF_00229]	Definition of API function CanIf_GetControllerMode
[SWS_CANIF_00230]	Definition of API function CanIf_ReadRxNotifStatus
[SWS_CANIF_00281]	
[SWS_CANIF_00286]	
[SWS_CANIF_00294]	Definition of optional interfaces requested by module CanIf
[SWS_CANIF_00297]	
[SWS_CANIF_00381]	

▽

△

Number	Heading
[SWS_CANIF_00382]	
[SWS_CANIF_00383]	
[SWS_CANIF_00389]	
[SWS_CANIF_00390]	
[SWS_CANIF_00489]	
[SWS_CANIF_00661]	
[SWS_CANIF_00665]	
[SWS_CANIF_00666]	
[SWS_CANIF_00677]	
[SWS_CANIF_00711]	
[SWS_CANIF_00739]	
[SWS_CANIF_00851]	
[SWS_CANIF_00877]	
[SWS_CANIF_00881]	
[SWS_CANIF_00885]	
[SWS_CANIF_00893]	
[SWS_CANIF_00894]	
[SWS_CANIF_00900]	
[SWS_CANIF_00903]	
[SWS_CANIF_00904]	
[SWS_CANIF_00905]	
[SWS_CANIF_00906]	
[SWS_CANIF_00911]	
[SWS_CANIF_00913]	
[SWS_CANIF_00915]	
[SWS_CANIF_00916]	
[SWS_CANIF_00917]	
[SWS_CANIF_00918]	
[SWS_CANIF_00919]	
[SWS_CANIF_00920]	
[SWS_CANIF_00921]	
[SWS_CANIF_00937]	
[SWS_CANIF_00938]	
[SWS_CANIF_00939]	
[SWS_CANIF_00940]	
[SWS_CANIF_00941]	
[SWS_CANIF_00942]	
[SWS_CANIF_00943]	

▽

△

Number	Heading
[SWS_CANIF_00944]	
[SWS_CANIF_00945]	
[SWS_CANIF_00946]	
[SWS_CANIF_00947]	
[SWS_CANIF_00948]	
[SWS_CANIF_00949]	
[SWS_CANIF_00950]	
[SWS_CANIF_00951]	
[SWS_CANIF_00952]	
[SWS_CANIF_00953]	
[SWS_CANIF_00954]	
[SWS_CANIF_00955]	
[SWS_CANIF_00956]	
[SWS_CANIF_00957]	
[SWS_CANIF_00967]	Support of meta data type `TIMETUPLE_TYPE_PTR` at reception indication
[SWS_CANIF_91001]	Definition of API function CanIf_GetControllerErrorState
[SWS_CANIF_91002]	Definition of API function CanIf_DeInit
[SWS_CANIF_91008]	Definition of API function CanIf_ControllerErrorStatePassive
[SWS_CANIF_91009]	Definition of API function CanIf_ErrorNotification
[SWS_CANIF_91010]	Security events for CanIf
[SWS_CANIF_91011]	Definition of API function CanIf_EnableEgressTimeStamp
[SWS_CANIF_91012]	Definition of API function CanIf_GetEgressTimeStamp
[SWS_CANIF_91013]	Definition of API function CanIf_GetIngressTimeStamp
[SWS_CANIF_91014]	Definition of API function CanIf_GetCurrentTime
[SWS_CANIF_91018]	Definition of API function CanIf_XLReportErrorState
[SWS_CANIF_92000]	Security event context data definition: SEV_CAN_TX_ERROR_DETECTED
[SWS_CANIF_92001]	Security event context data definition: SEV_CAN_RX_ERROR_DETECTED
[SWS_CANIF_92002]	Security event context data definition: SEV_CAN_ERRORSTATE_PASSIVE
[SWS_CANIF_92003]	Security event context data definition: SEV_CAN_ERRORSTATE_BUSOFF

Table B.11: Changed Specification Items in R26-11

B.4.3 Deleted Specification Items in R26-11

Number	Heading
[ECUC_CanIf_00613]	Definition of EcucBooleanParamDef CanIfVersionInfoApi

▽

△

Number	Heading
[SWS_CANIF_00158]	Definition of API function CanIf_GetVersionInfo

Table B.12: Deleted Specification Items in R26-11

B.4.4 Added Constraints in R26-11

Number	Heading
[SWS_CANIF_CONSTR_00015]	Same configuration of PDUs that belong to the same CanIfCtrlCfg for PduBuffer Alignment
[SWS_CANIF_CONSTR_00016]	Memory address alignment of the used hardware
[SWS_CANIF_CONSTR_00017]	Pdu.KeepLocalPduBuffer applicable to CAN XL PDUs only

Table B.13: Added Constraints in R26-11

B.4.5 Changed Constraints in R26-11

Number	Heading
[SWS_CANIF_CONSTR_00001]	
[SWS_CANIF_CONSTR_00002]	
[SWS_CANIF_CONSTR_00003]	
[SWS_CANIF_CONSTR_00004]	
[SWS_CANIF_CONSTR_00005]	
[SWS_CANIF_CONSTR_00006]	

▽

△

Number	Heading
[SWS_CANIF_CONSTR_00007]	
[SWS_CANIF_CONSTR_00008]	
[SWS_CANIF_CONSTR_00009]	
[SWS_CANIF_CONSTR_00010]	
[SWS_CANIF_CONSTR_00011]	
[SWS_CANIF_CONSTR_00012]	
[SWS_CANIF_CONSTR_00013]	
[SWS_CANIF_CONSTR_00014]	

Table B.14: Changed Constraints in R26-11

B.4.6 Deleted Constraints in R26-11

none