

<b>Document Title</b>	Specification of CAN Interface
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	12

<b>Document Status</b>	published
<b>Part of AUTOSAR Standard</b>	Classic Platform
<b>Part of Standard Release</b>	R22-11

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2022-11-24	R22-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• CAN XL Support</li> <li>• Editorial changes</li> </ul>
2021-11-25	R21-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Support hardware-based timestamping</li> <li>• Minor corrections/clarifications</li> <li>• Editorial changes</li> </ul>
2020-11-30	R20-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Introduction of Security Events</li> <li>• Rework of Error classification chapters</li> <li>• Removal of pretended networking</li> <li>• Minor corrections/clarifications</li> </ul>
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Update reference to ISO 11898-1:2015</li> <li>• Minor corrections</li> <li>• Editorial changes</li> <li>• Changed Document Status from Final to published</li> </ul>
2018-10-31	4.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• BusMirroring (CONC_634)</li> <li>• Receive Data Length Check per Pdu</li> <li>• Remove dummy implementations for Cancel Transmit APIs</li> <li>• Header File Cleanup</li> </ul>

2017-12-08	4.3.1	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Introduction of Runtime errors</li> <li>• Replace Can_ReturnType with Std_ReturnType overlay</li> <li>• Minor corrections</li> <li>• Editorial changes</li> </ul>
2016-11-30	4.3.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Remove CCMSM</li> <li>• Rework MetaData handling</li> <li>• Reliable TxConfirmation</li> <li>• Error Active/Passive State API</li> </ul>
2015-07-31	4.2.2	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Clarified wakeup, buffering, transmit, and variants</li> <li>• Removed deprecated APIs</li> <li>• Editorial changes</li> </ul>
2014-10-31	4.2.1	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Full CAN FD Support</li> <li>• Global Time Synchronization over CAN</li> <li>• Removed CanIf_CancelTxConfirmation</li> <li>• Small improvements</li> </ul>
2014-03-31	4.1.3	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Removed BSW Exclusive areas</li> <li>• Set ICOM support to optional</li> <li>• Can_IdType handling</li> <li>• Small improvements</li> </ul>
2013-10-31	4.1.2	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Restricted PDU mode changes</li> <li>• Removed critical section handling description in <a href="#">chapter 9</a></li> <li>• Set CanIfInitRefCfgSet obsolete</li> <li>• Pretended Networking section</li> <li>• Small improvements</li> </ul>
2013-03-15	4.1.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• CAN FD (without DLC extension)</li> <li>• Pretended Networking (ICOM)</li> <li>• Heavy Duty Vehicle (J1939) support</li> <li>• PduModes and PnTxFilter for clean wake-up</li> <li>• Relation between PDUs &amp; HOHs</li> <li>• Post-build loadable concept</li> </ul>
2011-12-22	4.0.3	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Partial Networking Support</li> <li>• Improved Transmit Buffering</li> <li>• Improved Error Detection</li> </ul>

2009-12-18	4.0.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Updated chapters "Version Checking" and "Published Information"</li> <li>• Multiple CAN IDs could optionally be assigned to one I-PDU</li> <li>• Wake-up validation optionally only via NM PDUs</li> <li>• Asynch. mode indication call-backs instead of synch. mode changes</li> <li>• No automatic PDU channel mode change when CC mode changes</li> <li>• TxConfirmation state entered for BusOff Recovery</li> <li>• WakeupSourceRefIn and WakeupSourceRefOut</li> <li>• PduInfoPtr instead of SduDataPtr</li> <li>• Introduction of Can_GeneralTypes.h and Can_HwHandleType</li> <li>• Transceiver types of chapter 8. shifted to transceiver SWS</li> </ul>
2010-02-02	3.1.4	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• HOH definition</li> <li>• abstracted ControllerId and TransceiverId</li> <li>• No changing of baudrate via CanIf and CanIf_ControllerInit</li> <li>• Dispatcher adapted because of CDD</li> <li>• TxBuffering: only one buffer per L-PDU</li> <li>• Wake up mechanism adapted to environment behavior (network -&gt; controller/transceiver; wakeupSource)</li> <li>• Mode changes made asynchronous</li> <li>• no complete state machine in CanIf, just buffered states per controller</li> <li>• Legal disclaimer revised</li> </ul>
2008-08-13	3.1.1	AUTOSAR Administration	Legal disclaimer revised
2008-02-01	3.0.2	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Replaced chapter 10 content with generated tables from AUTOSAR MetaModel.</li> </ul>

2008-02-01	3.0.2	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Interface abstraction: network related interface changed into a controller related one</li> <li>• Wakeup mechanism completely reworked, APIs added &amp; changed for Wakeup</li> <li>• Initialization changed (flat initialization)</li> <li>• Scheduled main functions skipped due to changed BSW Scheduler responsibility</li> <li>• Document meta information extended</li> <li>• Small layout adaptations made</li> </ul>
2007-12-21	3.0.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Header file structure changed</li> <li>• Support of mixed mode operation (Standard CAN &amp; Extended CAN in parallel on one network) added</li> <li>• Support of CAN Transceiver API &lt;User&gt;_DlcErrorNotification deleted</li> <li>• Pre-compile/Link-Time/Post-Built definition for configuration parameters partly changed</li> <li>• Re-entrant interface call allowed for certain APIs</li> <li>• Support of AUTOSAR BSW Scheduler added</li> <li>• Support of memory mapping added</li> <li>• Configuration container structure reworked</li> <li>• Various of clarification extensions and corrections</li> </ul>
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.

## Contents

1	Introduction and functional overview	11
2	Acronyms and Abbreviations	14
3	Related documentation	16
3.1	Input documents & related standards and norms	16
3.2	Related specification	17
4	Constraints and assumptions	18
4.1	Limitations	18
4.2	Applicability to car domains	18
5	Dependencies to other modules	19
5.1	Upper Protocol Layers	20
5.2	Initialization: Ecu State Manager	20
5.3	Mode Control: CAN State Manager	20
5.4	Lower layers: CAN Driver	20
5.5	Lower layers: CAN Transceiver Driver	21
5.6	Configuration	22
5.7	File structure	23
5.7.1	Code file structure	23
5.7.2	Header file structure	23
6	Requirements Tracing	24
7	Functional specification	31
7.1	General Functionality	31
7.2	Hardware object handles	32
7.3	Static L-PDUs	34
7.4	Dynamic L-PDUs	35
7.4.1	Dynamic Transmit L-PDUs	36
7.4.2	Dynamic receive L-PDUs	37
7.5	Physical channel view	37
7.6	CAN Hardware Unit	39
7.7	BasicCAN and FullCAN reception	40
7.8	Initialization	42
7.9	Transmit request	42
7.10	Transmit data flow	43
7.11	Transmit buffering	44
7.11.1	General behavior	45
7.11.2	Buffer characteristics	46
7.11.2.1	Storage of L-PDUs in the transmit L-PDU buffer	46
7.11.2.2	Clearance of transmit L-PDU buffers	48
7.11.2.3	Initialization of transmit L-PDU buffers	48
7.11.3	Data integrity of transmit L-PDU buffers	48

7.12	Transmit confirmation	49
7.13	Receive data flow	49
7.14	Receive indication	51
7.15	Read received data	53
7.16	Read Tx/Rx notification status	54
7.17	Data integrity	54
7.18	CAN Controller Mode	55
7.18.1	General Functionality	55
7.18.2	CAN Controller Operation Modes	56
7.18.3	Controller Mode Transitions	57
7.18.4	Wake-up	57
7.18.4.1	Wake-up detection	58
7.18.4.2	Wake-up Validation	58
7.19	PDU channel mode control	59
7.19.1	PDU channel groups	59
7.19.2	PDU channel modes	60
7.19.2.1	CANIF_OFFLINE	61
7.19.2.2	CANIF_ONLINE	62
7.19.2.3	CANIF_OFFLINE_ACTIVE	63
7.20	Software receive filter	63
7.20.1	Software filtering concept	64
7.20.2	Software filter algorithms	66
7.21	Data Length Check	66
7.22	L-SDU dispatcher to upper layers	67
7.23	Polling mode	67
7.24	Multiple CAN Driver support	67
7.24.1	Transmit requests by using multiple CAN Drivers	68
7.24.2	Notification mechanism using multiple CAN Drivers	69
7.25	Partial Networking	71
7.26	CAN FD Support	72
7.27	CAN XL Support	73
7.27.1	Dynamic Control of CAN XL Attributes	74
7.28	Security Events	74
7.29	Error classification	76
7.29.1	Development Errors	76
7.29.2	Runtime Errors	77
7.29.3	Transient Faults	77
7.29.4	Production Errors	77
7.29.5	Extended Production Errors	77
8	API specification	78
8.1	Imported types	78
8.2	Type definitions	78
8.2.1	CanIf_ConfigType	78
8.2.2	CanIf_PduModeType	79
8.2.3	CanIf_NotifStatusType	80

8.3	Function definitions	80
8.3.1	Canlf_Init	80
8.3.2	Canlf_DelInit	81
8.3.3	Canlf_SetControllerMode	81
8.3.4	Canlf_GetControllerMode	82
8.3.5	Canlf_GetControllerErrorState	83
8.3.6	Canlf_Transmit	84
8.3.7	Canlf_ReadRxPduData	87
8.3.8	Canlf_ReadTxNotifStatus	88
8.3.9	Canlf_ReadRxNotifStatus	89
8.3.10	Canlf_SetPduMode	90
8.3.11	Canlf_GetPduMode	91
8.3.12	Canlf_GetVersionInfo	91
8.3.13	Canlf_SetDynamicTxId	92
8.3.14	Canlf_SetTrcvMode	93
8.3.15	Canlf_GetTrcvMode	94
8.3.16	Canlf_GetTrcvWakeupReason	95
8.3.17	Canlf_SetTrcvWakeupMode	97
8.3.18	Canlf_CheckWakeup	98
8.3.19	Canlf_CheckValidation	99
8.3.20	Canlf_GetTxConfirmationState	100
8.3.21	Canlf_ClearTrcvWufFlag	101
8.3.22	Canlf_CheckTrcvWakeFlag	101
8.3.23	Canlf_SetBaudrate	102
8.3.24	Canlf_GetControllerRxErrorCounter	103
8.3.25	Canlf_GetControllerTxErrorCounter	104
8.3.26	Canlf_EnableBusMirroring	104
8.3.27	Canlf_GetCurrentTime	105
8.3.28	Canlf_EnableEgressTimeStamp	106
8.3.29	Canlf_GetEgressTimeStamp	107
8.3.30	Canlf_GetIngressTimeStamp	107
8.4	Callback notifications	108
8.4.1	Canlf_TriggerTransmit	108
8.4.2	Canlf_TxConfirmation	109
8.4.3	Canlf_RxIndication	110
8.4.4	Canlf_XLRxIndication	112
8.4.5	Canlf_ControllerBusOff	113
8.4.6	Canlf_ConfirmPnAvailability	114
8.4.7	Canlf_ClearTrcvWufFlagIndication	115
8.4.8	Canlf_CheckTrcvWakeFlagIndication	116
8.4.9	Canlf_ControllerModeIndication	117
8.4.10	Canlf_TrvcModeIndication	117
8.4.11	Canlf_ControllerErrorStatePassive	119
8.4.12	Canlf_ErrorNotification	119
8.5	Scheduled functions	120
8.6	Expected interfaces	120



8.6.1	Mandatory interfaces	120
8.6.2	Optional interfaces	121
8.6.3	Configurable interfaces	123
8.6.3.1	<User_TriggerTransmit>	124
8.6.3.2	<User_TxConfirmation>	125
8.6.3.3	<User_RxIndication>	127
8.6.3.4	<User_ValidateWakeupEvent>	128
8.6.3.5	<User_ControllerBusOff>	130
8.6.3.6	<User_ConfirmPnAvailability>	131
8.6.3.7	<User_ClearTrcvWufFlagIndication>	132
8.6.3.8	<User_CheckTrcvWakeFlagIndication>	133
8.6.3.9	<User_ControllerModeIndication>	134
8.6.3.10	<User_TrvcModeIndication>	135
9	Sequence diagrams	138
9.1	Transmit request (single CAN Driver)	138
9.2	Transmit request (multiple CAN Drivers)	139
9.3	Transmit confirmation (interrupt mode)	141
9.4	Transmit confirmation (polling mode)	142
9.5	Transmit confirmation (with buffering)	143
9.6	Trigger Transmit Request	144
9.7	Receive indication (interrupt mode)	146
9.8	Receive indication (polling mode)	148
9.9	Read received data	150
9.10	Start CAN network	152
9.11	BusOff notification	154
9.12	BusOff recovery	155
10	Configuration specification	157
10.1	Containers and configuration parameters	157
10.1.1	CanIf	158
10.1.2	CanIfPrivateCfg	160
10.1.3	CanIfPublicCfg	162
10.1.4	CanIfInitCfg	172
10.1.5	CanIfTxPduCfg	175
10.1.6	CanIfTxPduXLParams	183
10.1.7	CanIfRxPduCfg	185
10.1.8	CanIfRxPduXLParams	192
10.1.9	CanIfRxPduCanIdRange	194
10.1.10	CanIfDispatchCfg	195
10.1.11	CanIfCtrlCfg	204
10.1.12	CanIfCtrlDrvCfg	206
10.1.13	CanIfTrcvDrvCfg	207
10.1.14	CanIfTrcvCfg	208
10.1.15	CanIfInitHohCfg	210
10.1.16	CanIfHthCfg	211
10.1.17	CanIfHrhCfg	212

10.1.18	CanIfHrhRangeCfg	214
10.1.19	CanIfBufferCfg	217
10.1.20	CanIfSecurityEventRefs	219
10.2	Configuration constraints	221
A	Not applicable requirements	224

## 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 Controller-s/CAN Transceivers can be controlled by the CAN State Managers module based on a physical CAN channel related view.

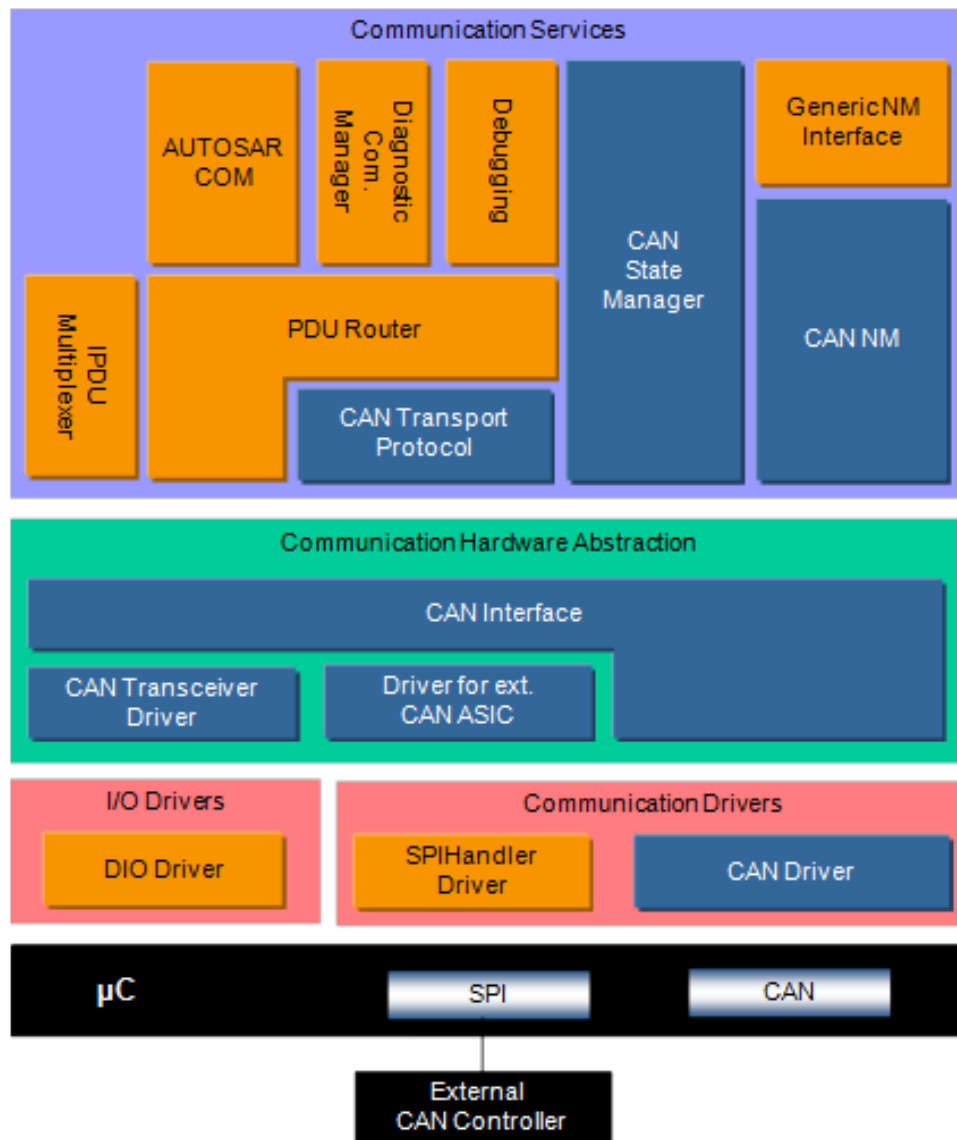


Figure 1.1: AUTOSAR CAN Layer Model (see [7])

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 [8, 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
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: <ul style="list-style-type: none"> <li>• Participating nodes</li> <li>• Definition of all CAN PDUs (Identifier, Data Length)</li> <li>• Source and Sinks for PDUs</li> </ul>
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.
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.
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

#### References

- [1] Specification of CAN Driver  
AUTOSAR\_SWS\_CANDriver
- [2] Specification of CAN Transceiver Driver  
AUTOSAR\_SWS\_CANTransceiverDriver
- [3] Specification of CAN State Manager  
AUTOSAR\_SWS\_CANStateManager
- [4] Specification of CAN Network Management  
AUTOSAR\_SWS\_CANNetworkManagement
- [5] Specification of CAN Transport Layer  
AUTOSAR\_SWS\_CANTransportLayer
- [6] Specification of PDU Router  
AUTOSAR\_SWS\_PDURouter
- [7] Layered Software Architecture  
AUTOSAR\_EXP\_LayeredSoftwareArchitecture
- [8] Glossary  
AUTOSAR\_TR\_Glossary
- [9] General Specification of Basic Software Modules  
AUTOSAR\_SWS\_BSWGeneral
- [10] General Requirements on Basic Software Modules  
AUTOSAR\_SRS\_BSWGeneral
- [11] Requirements on CAN  
AUTOSAR\_SRS\_CAN
- [12] ISO 11898-1:2015 – Road vehicles – Controller area network (CAN)
- [13] Specification of ECU State Manager  
AUTOSAR\_SWS\_ECUStateManager
- [14] 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>
- [15] 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>



[16] Specification of ECU Configuration  
AUTOSAR\_TPS\_ECUConfiguration

### **3.2 Related specification**

AUTOSAR provides a General Specification on Basic Software modules [9, 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.

### 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](#)).

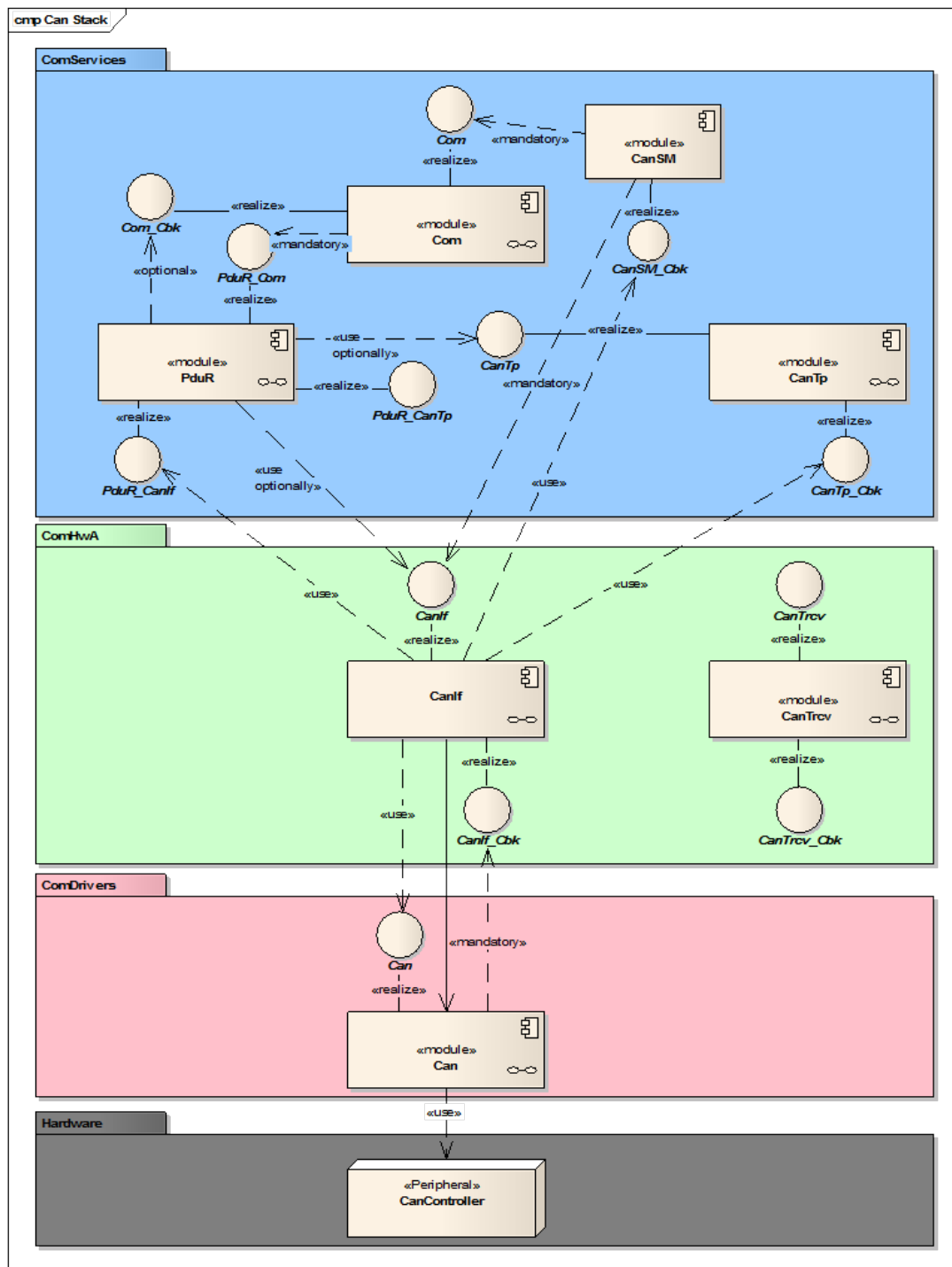


Figure 5.1: CANIF dependencies in AUTOSAR BSW

## 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 [section 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 [section 7.24 “Multiple CAN Driver support”](#). Reentrancy of callback functions is specified in [section 8.4](#).

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 [section 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 [section 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 [section 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).] (SRS\_Can\_01172)

## 6 Requirements Tracing

The following tables references the requirements specified in [10] as well as [11] and links to the fulfillment of these. Please note that if column 'Satisfied by' is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_Ids_00810]	Basic SW security events	<a href="#">[SWS_CANIF_00913]</a> <a href="#">[SWS_CANIF_00915]</a> <a href="#">[SWS_CANIF_00916]</a> <a href="#">[SWS_CANIF_00917]</a> <a href="#">[SWS_CANIF_00918]</a> <a href="#">[SWS_CANIF_00919]</a> <a href="#">[SWS_CANIF_00920]</a> <a href="#">[SWS_CANIF_00921]</a> <a href="#">[SWS_CANIF_91008]</a> <a href="#">[SWS_CANIF_91009]</a> <a href="#">[SWS_CANIF_91010]</a>
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	<a href="#">[SWS_CANIF_00001]</a>
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	<a href="#">[SWS_CANIF_NA_00999]</a>
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	<a href="#">[SWS_CANIF_NA_00999]</a>



[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_CANIF_NA_00999]
[SRS_BSW_00307]	Global variables naming convention	[SWS_CANIF_NA_00999]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_CANIF_NA_00999]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_CANIF_NA_00999]
[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_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_CANIF_00135]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_CANIF_NA_00999]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_CANIF_NA_00999]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_CANIF_NA_00999]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_CANIF_91002] [SWS_CANIF_NA_00999]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_CANIF_NA_00999]
[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 integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_CANIF_00142]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_CANIF_00001]
[SRS_BSW_00361]	No description	[SWS_CANIF_00142]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_CANIF_NA_00999]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_CANIF_NA_00999]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_CANIF_00001]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_CANIF_00158]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_CANIF_00158]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_CANIF_00001]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_CANIF_NA_00999]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the Dem is fully operational.	[SWS_CANIF_NA_00999]

[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_CANIF_NA_00999]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_CANIF_NA_00999]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_CANIF_NA_00999]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_CANIF_NA_00999]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_CANIF_NA_00999]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_CANIF_NA_00999]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_CANIF_NA_00999]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_CANIF_NA_00999]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_CANIF_NA_00999]
[SRS_Can_01001]	The CAN Interface implementation and interface shall be independent from underlying CAN Controller and CAN Transceiver	[SWS_CANIF_00023]
[SRS_Can_01003]	The appropriate higher communication stack shall be notified by the CAN Interface about an occurred reception	[SWS_CANIF_00012]
[SRS_Can_01005]	The CAN Interface shall perform a check for correct DLC of received PDUs	[SWS_CANIF_00026]
[SRS_Can_01008]	The CAN Interface shall provide a transmission request service	[SWS_CANIF_00005]
[SRS_Can_01009]	The CAN Interface shall provide a transmission confirmation dispatcher	[SWS_CANIF_00007]
[SRS_Can_01011]	The CAN Interface shall provide a transmit buffer	[SWS_CANIF_00068]
[SRS_Can_01014]	The CAN State Manager shall offer a network configuration independent interface for upper layers	[SWS_CANIF_NA_00999]

[SRS_Can_01018]	The CAN Interface shall allow the configuration of its software reception filter Pre-Compile-Time as well as Link-Time and Post-Build-Time	[SWS_CANIF_00030]
[SRS_Can_01020]	The TX-Buffer shall be statically configurable	[SWS_CANIF_00063]
[SRS_Can_01021]	CAN The CAN Interface shall implement an interface for initialization	[SWS_CANIF_00001]
[SRS_Can_01022]	The CAN Interface shall support the selection of configuration sets	[SWS_CANIF_00001]
[SRS_Can_01027]	The CAN Interface shall provide a service to change the CAN Controller mode.	[SWS_CANIF_00003]
[SRS_Can_01028]	The CAN Interface shall provide a service to query the CAN controller state	[SWS_CANIF_00229]
[SRS_Can_01029]	The CAN Interface shall report bus-off state of a device to an upper layer	[SWS_CANIF_00014]
[SRS_Can_01114]	Data Consistency of L-PDUs to transmit shall be guaranteed	[SWS_CANIF_00033]
[SRS_Can_01125]	The CAN stack shall ensure not to lose messages in receive direction	[SWS_CANIF_00194]
[SRS_Can_01126]	The CAN stack shall be able to produce 100% bus load	[SWS_CANIF_00381] [SWS_CANIF_00382] [SWS_CANIF_00881]
[SRS_Can_01129]	The CAN Interface module shall provide a procedural interface to read out data of single CAN messages by upper layers (Polling mechanism)	[SWS_CANIF_00194]
[SRS_Can_01130]	Receive Status Interface of CAN Interface	[SWS_CANIF_00202] [SWS_CANIF_00230]
[SRS_Can_01131]	The CAN Interface module shall provide the possibility to have polling and callback notification mechanism in parallel	[SWS_CANIF_00230]
[SRS_Can_01136]	The CAN Interface module shall provide a service to check for validation of a CAN wake-up event	[SWS_CANIF_00179]
[SRS_Can_01139]	The CAN Interface and Driver shall offer a CAN Controller specific interface for initialization	[SWS_CANIF_NA_00999]
[SRS_Can_01140]	The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers	[SWS_CANIF_00281] [SWS_CANIF_00877]
[SRS_Can_01141]	The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers at same time on one network	[SWS_CANIF_00243] [SWS_CANIF_00877]

[SRS_Can_01151]	The CAN Interface shall provide a service to check for a CAN Wake-up event.	[SWS_CANIF_00286]
[SRS_Can_01162]	The CAN Interface shall support classic CAN and CAN FD frames	[SWS_CANIF_00162] [SWS_CANIF_00677] [SWS_CANIF_00877] [SWS_CANIF_00893] [SWS_CANIF_00939]
[SRS_Can_01168]	The CAN Interface shall implement an interface for de-initialization	[SWS_CANIF_91002]
[SRS_Can_01169]	The CAN interface shall provide a function to return the current CAN controller error state	[SWS_CANIF_91001]
[SRS_Can_01172]	The CAN Interface shall provide a function to provide received and transmitted frames to the Bus Mirroring	[SWS_CANIF_00903] [SWS_CANIF_00904] [SWS_CANIF_00905] [SWS_CANIF_00906] [SWS_CANIF_00911]
[SRS_Can_01173]	No description	[SWS_CANIF_00750]
[SRS_Can_01174]	No description	[SWS_CANIF_00747] [SWS_CANIF_00748] [SWS_CANIF_00750]
[SRS_Can_01175]	No description	[SWS_CANIF_00290]
[SRS_Can_01181]	If partial networking is used, the ECU shall secure that the first message on the bus is the wakeup frame.	[SWS_CANIF_91011] [SWS_CANIF_91012] [SWS_CANIF_91013] [SWS_CANIF_91014]
[SRS_Can_02003]	The CAN Interface shall support CAN XL frames	[SWS_CANIF_00162] [SWS_CANIF_00677] [SWS_CANIF_00893] [SWS_CANIF_00937] [SWS_CANIF_00938] [SWS_CANIF_00939] [SWS_CANIF_00940] [SWS_CANIF_00941] [SWS_CANIF_00942] [SWS_CANIF_00943] [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_CONSTR_00001] [SWS_CANIF_CONSTR_00002] [SWS_CANIF_CONSTR_00003] [SWS_CANIF_CONSTR_00004] [SWS_CANIF_CONSTR_00005] [SWS_CANIF_CONSTR_00006] [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]
--	--	--

## 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.] (*SRS\_Can\_01001*)

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.]()

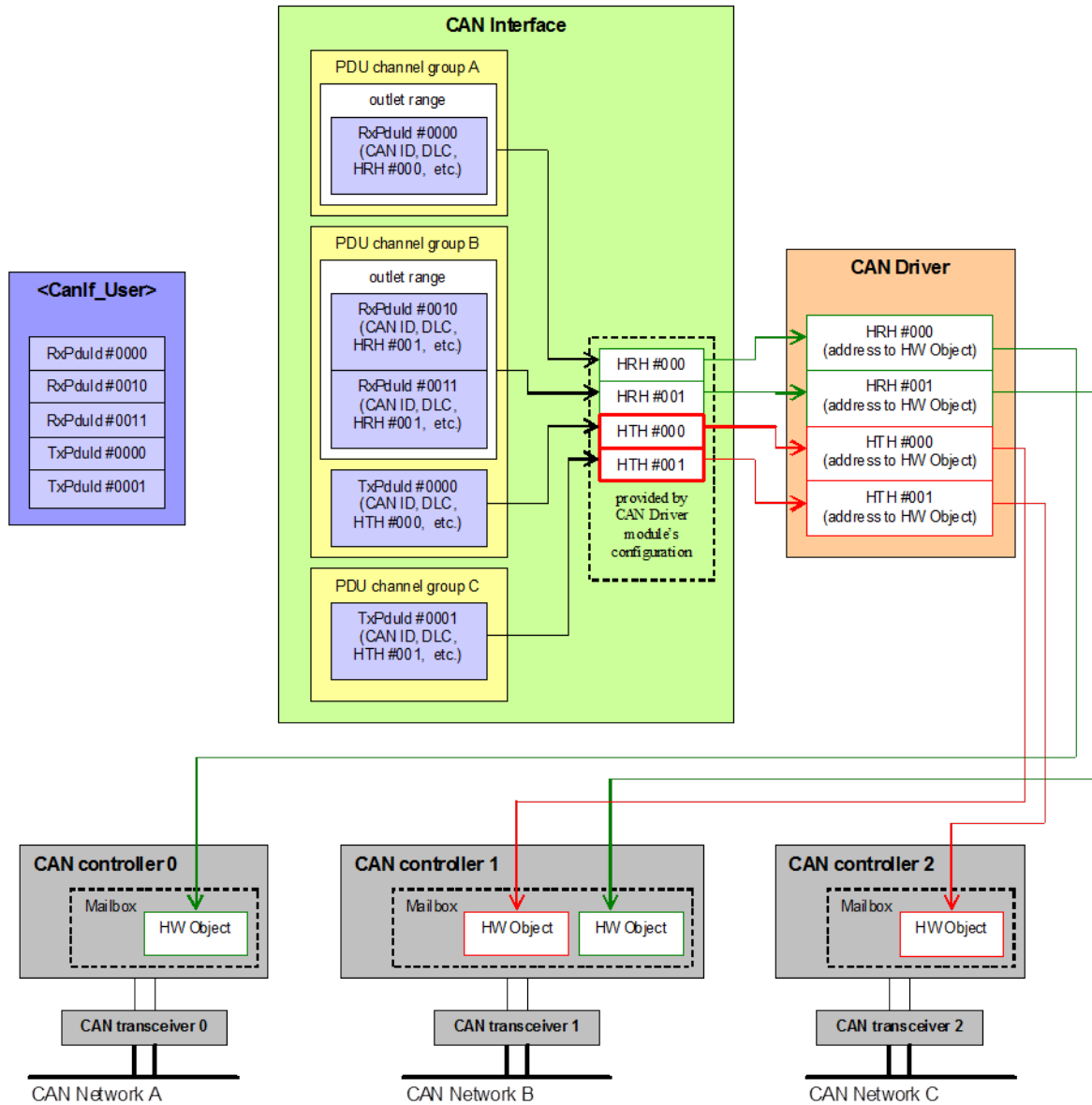
**[SWS\_CANIF\_00663]** [If the **HRH** references a reception unit configured for *BasicCAN reception*, software filtering shall be enabled in `CanIf`.]()

**[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.



All arrows within this picture are references and no communication directions of sth. else.

**Descriptions:**  
 Outlet range= Range of Rx L-PDUs which will be passed  
 Mailbox = CAN RAM structure  
 HWObject = CAN RAM structure that contains (CanId, DLC, data)  
 HRH = a abstract reference to the CAN RAM structure  
 Transmit path is coloured red  
 Receive path is coloured green.

**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 <b>CanIfPrivateSoftwareFilterType</b>	<b>CAN Identifier</b> ( <b>CanId</b> ) <b>CanIfTxPduCanId</b> , range of <b>CanIds</b> per <b>PDU</b> (see <b>CanIfRxPduCanIdRange</b> ), <b>CanIfRxPduCanId</b> , <b>CanIfRxPduCanIdMask</b>
Direction of <b>L-PDU</b> (Tx, Rx) <b>CanIfTxPduId</b> , <b>CanIfRxPduId</b> )	Type of <b>CAN Identifier</b> ( <i>StandardCAN</i> , <i>ExtendedCAN</i> ) referenced from <b>CanDrv</b> via <b>CanIfHthIdSymRef</b> , <b>CanIfHrhIdSymRef</b>
<b>HTH/HRH</b> of the <b>CAN Controller</b>	Data Length and Data Length Code ( <b>DLC</b> ) <b>CanIfRxPduDataLength</b>
Target ID for the corresponding upper layer <b>CanIfTxPduUserTxConfirmationUL</b> , <b>CanIfRxPduUserRxIndicationUL</b>	Reference to the PDU data (see [1, Specification of CAN Driver])
Type of <b>Transmit L-PDU</b> (STATIC, DYNAMIC) <b>CanIfTxPduType</b>	
Type of <b>Tx/Rx L-PDU</b> ( <i>FullCAN</i> , <i>BasicCAN</i> ) <b>CanIfHthIdSymRef</b> , <b>CanIfHrhIdSymRef</b>	

`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 section 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 [7, 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 subsection 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 usage of all other L-PDU elements are equal to normal static `Transmit L-PDUs`:

- The transmit confirmation notification `CanIfTxPduUserTxConfirmationUL` cannot be reconfigured as it belongs to the L-PDU.
- The Data Length and the pointer to the data buffer are both determined by the upper layer module at call of `CanIf_Transmit()`.

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` and the bits of the `Can_IdType` derived from `CanIfTxPduCanIdType` 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:  $(\text{CanIfTxPduCanId} \& \text{CanIfTxPduCanIdMask}) | (\text{dynamic ID parts} \& \sim \text{CanIfTxPduCanIdMask})$

**[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 *CanIfCtrlId*.]()

Example:

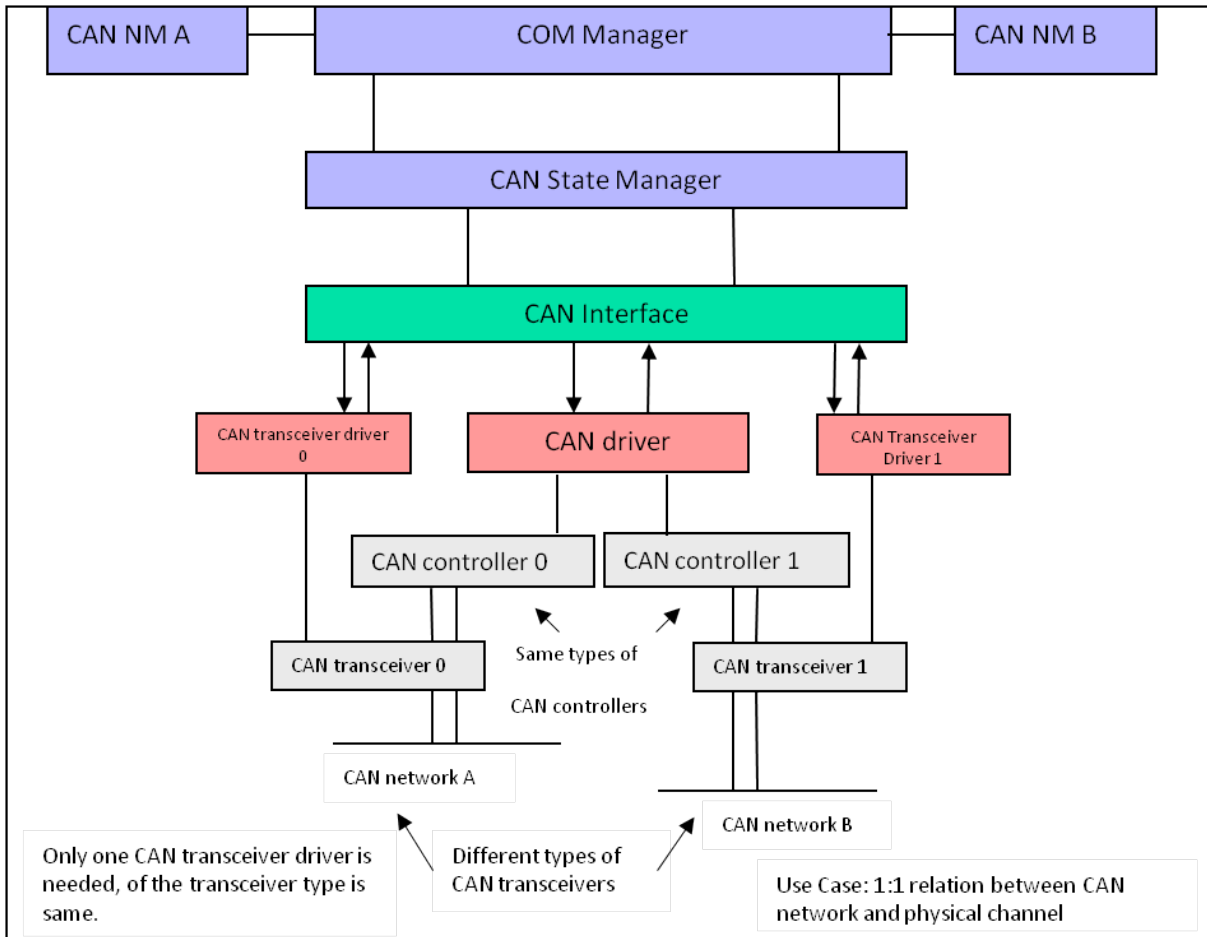
CanIf	CanDrv A	CanDrv B
ControllerId 0	Controller 0	
ControllerId 1	Controller 1	
ControllerId 2		Controller 0

**[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

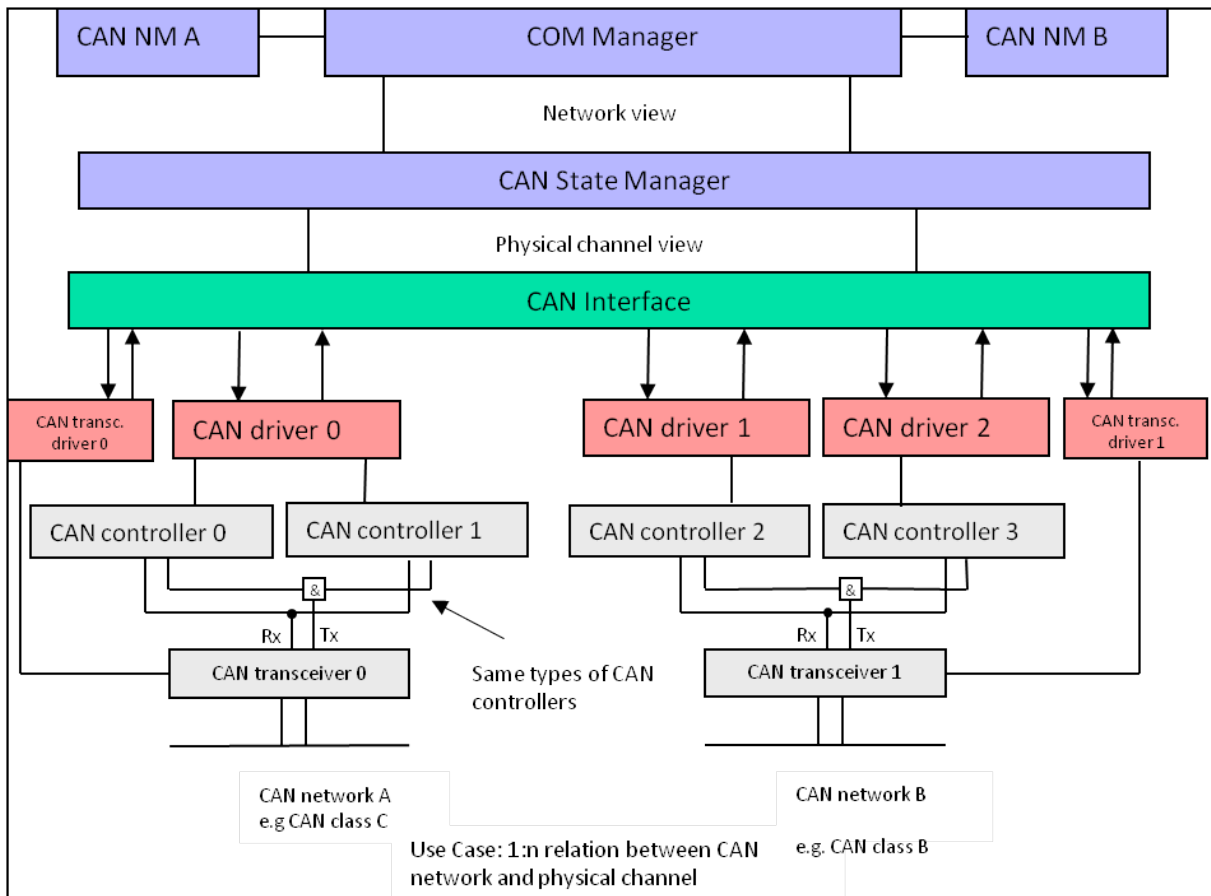
During the notification process the CanIf maps the original CAN Controller or CAN Transceiver parameter from the Driver module to the CanSm. This mapping is done as the referenced CAN Controller or CAN Transceiver parameters are configured with the abstracted CanIf parameters `ControllerId` or `TransceiverId`.



**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.



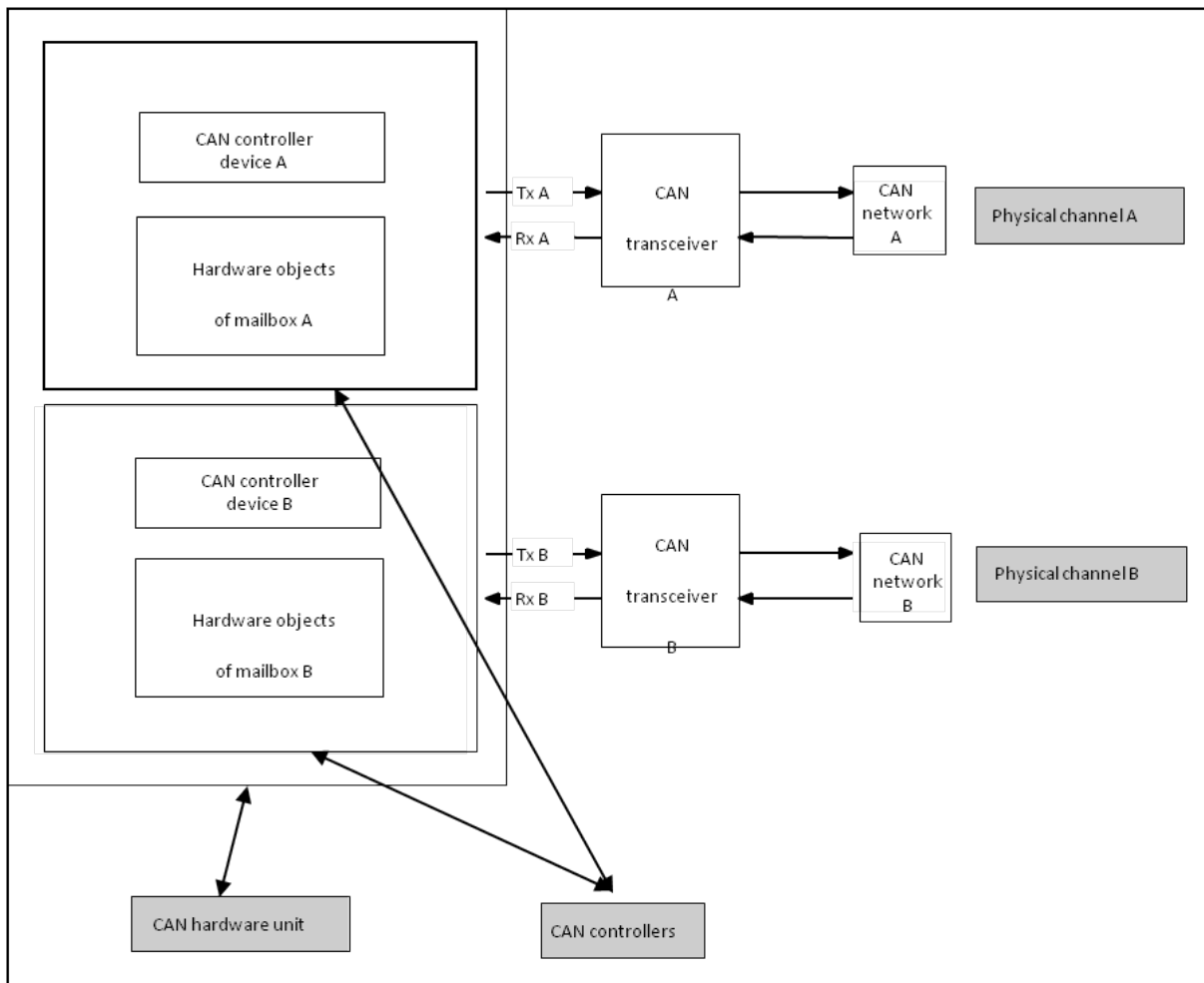
**Figure 7.3: Physical channel view definition example B**

## 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 [section 7.2 “Hardware object handles”](#) and [section 7.24 “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 [section 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 [section 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 a *L-PDU* (see *CanIf\_RxIndication()*), it shall perform the following comparisons to select the correct reception *L-SDU* configured in *CanIfRxPduCfg*:

- compare *CanIfRxPduCanId* with the passed *Mailbox->CanId* (*Can\_IdType*) excluding the two most significant bits
- compare *CanIfRxPduCanIdType* with the two most significant bits of the passed *Mailbox->CanId* (*Can\_IdType*)

]([SRS\\_Can\\_01140](#), [SRS\\_Can\\_01141](#), [SRS\\_Can\\_01162](#))

Basically, *CanIf* supports reception either of *Standard CAN IDs* or *Extended CAN IDs* on one *Physical CAN Channel* by the parameters *CanIfTxPduCanIdType* and *CanIfRxPduCanIdType*.

**[SWS\_CANIF\_00281]** [*CanIf* shall accept and handle *StandardCAN IDs* and *ExtendedCAN IDs* on the same *Physical Channel* (= mixed mode operation).]([SRS\\_Can\\_01140](#))

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 using configuration parameters *CanIfTxPduCanIdType* and *CanIfRxPduCanIdType*. In case of mixed

mode operation the software acceptance filter algorithm (see [section 7.20 “Software receive filter”](#)) must be able to deal with both type of `CanIds`.

[SWS\_CANIF\_00281] is an optional feature. This feature can be realized by different variants of implementations, no configuration options are available.

## 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 subsection 8.6.3).

## 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 subsection 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`.] (*SRS\_Can\_01126*)

Note for **[SWS\_CANIF\_00382]**: See subsection 7.19.2 “PDU channel modes”.

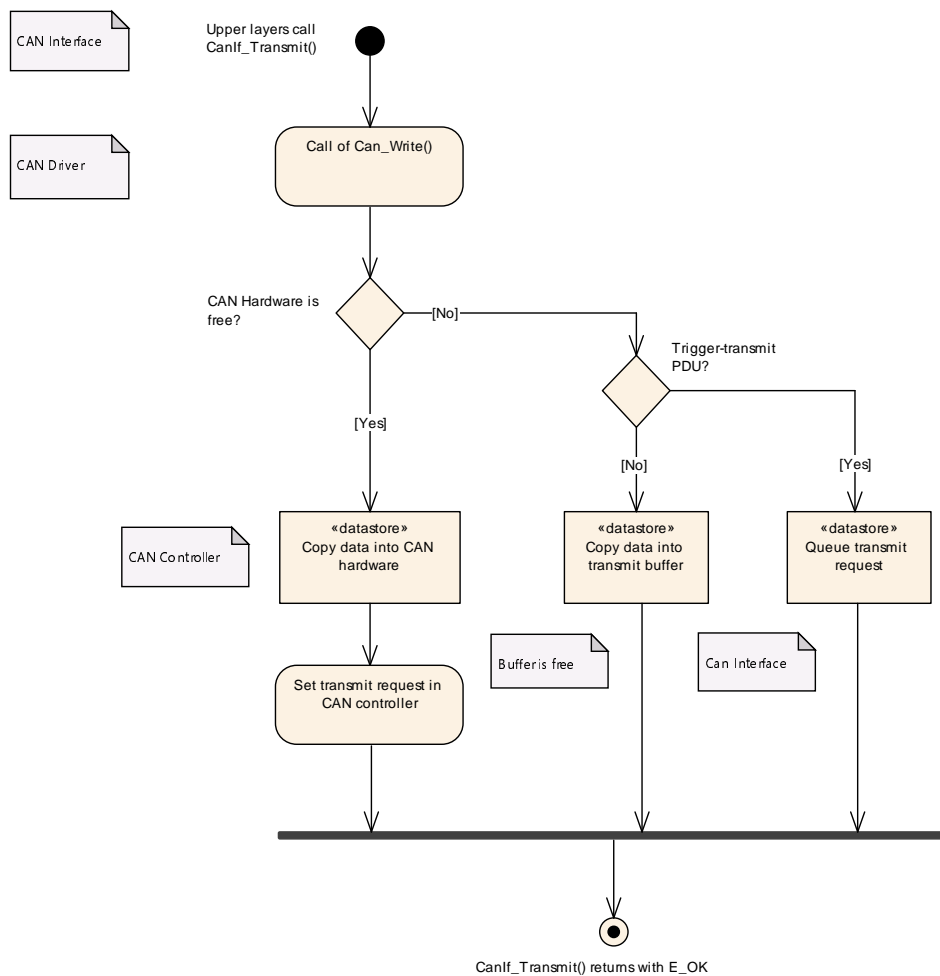
If the call of `Can_Write()` returns with `CAN_BUSY`, please refer to section 7.12 “Transmit confirmation” for further details.

## 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 `<User_RxIndication>()`. 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 CanTxPduId 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().] (SRS\_Can\_01172)

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 `<User_TxConfirmation>()`. 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 consists of one or multiple `CanIfBufferCfg` (see `CanIfBufferCfg`). Whereas each `CanIfBufferCfg` is assigned to one or multiple 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.] (*SRS\_Can\_01020*)

[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`.] (*SRS\_Can\_01126*)

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\_00895] [If the rejected data length exceeds the configured size, `CanIf` shall:

- buffer the configured amount of data and discard the rest
- and report runtime error code `CANIF_E_DATA_LENGTH_MISMATCH` to the `Det_ReportRuntimeError()` service of the DET.

]()

[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`.] (*SRS\_Can\_01126*)

**[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`.] (*SRS\_Can\_01011*)

Note: There is nothing to do for already stored `Transmit Requests` (see [\[SWS\\_CANIF\\_00068\]](#)) due to the fact the data will be caught 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 [12, 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.](SRS\_Can\_01114)

This may be realized by using exclusive areas defined within the *BSW Scheduler*. These exclusive areas can e.g. be 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`.] (SRS\_Can\_01172)

[SWS\_CANIF\_00383] [When callback notification `CanIf_TxConfirmation()` is called, `CanIf` shall identify the upper layer communication layer (see [SWS\_CANIF\_00414]), which is linked to the successfully transmitted `L-PDU`, and shall notify it about the performed transmission by call of `CanIf`'s transmit confirmation service `<User_TxConfirmation>(E_OK).()`

Note for [SWS\_CANIF\_00383]: See section 7.12 “Transmit confirmation”.

The callback service `<User_TxConfirmation>()` is implemented by the notified upper layer module.

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.

One transmit `L-PDU` can only be assigned to one single transmit confirmation callback service. Please refer to subsection 8.6.3.2 “`<User_TxConfirmation>`”.

[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 `CanIfPublicReadRxPduDataApi` is set to `TRUE` (refer to [section 7.15](#)). Tx buffering is addressed in [section 7.11](#) and dynamic L-PDUs are concerned in [section 7.4](#).

In case of a new reception of an L-PDU `CanDrv` calls `CanIf_RxIndication()` (refer to [\[SWS\\_CANIF\\_00006\]](#)) of `CanIf`. The access to the L-PDU specific data is organized by these parameters:

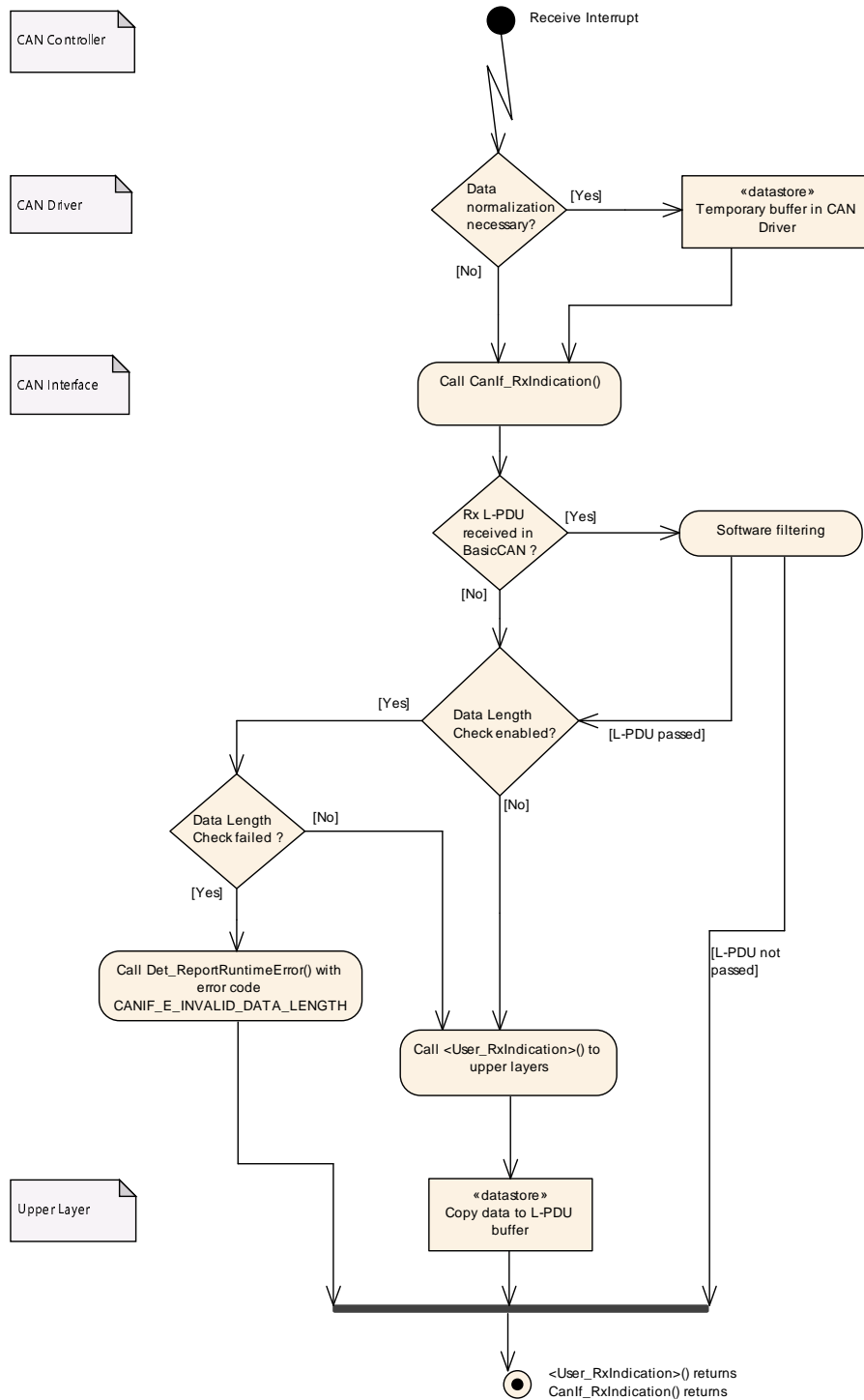
- Hardware Receive Handle (`HRH`)
- Received CAN Identifier (`CanId`)
- Received Data Length
- Reference to `Received L-PDU`

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()` 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()`.

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.

`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()` (see [SWS\_CANIF\_00006]) references in its parameters a newly received CAN L-PDU. If the function `CanIf_RxIndication()` 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 `<User_RxIndication>()` (see [subsection 8.6.3.3 “<User\\_RxIndication>”](#), [SWS\_CANIF\_00012]), 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\(\)](#).] ([SRS\\_Can\\_01172](#))

**[SWS\_CANIF\_00389]** [If the function [CanIf\\_RxIndication\(\)](#) 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 the receive indication for that call of [CanIf\\_RxIndication\(\)](#).] ()

Note for [SWS\_CANIF\_00389]: See [7.20](#).

**[SWS\_CANIF\_00390]** [If CanIf accepts an L-PDU received via [CanIf\\_RxIndication\(\)](#) 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 [section 7.21 “Data Length Check”](#).

**[SWS\_CANIF\_00297]** [If CanIf has accepted a 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 [ECUC\\_CanIf\\_00599](#)) to the static receive buffer, if configured for that L-PDU (see [SWS\_CANIF\_00198], [ECUC\\_CanIf\\_00600](#)).] ()

**[SWS\_CANIF\_00851]** [If MetaData item `CAN_ID_32` is configured for a received L-SDU, CanIf shall copy the PDU payload to the static receive buffer and the CAN ID to the `MetaDataItem` of type `CAN_ID_32`.] ()

**[SWS\_CANIF\_00056]** [If CanIf accepts a L-PDU received via [CanIf\\_RxIndication\(\)](#) during Data Length Check (see [SWS\_CANIF\_00390], [SWS\_CANIF\_00026]), CanIf shall identify if a target upper layer module was configured (see configuration description of [SWS\_CANIF\_00012], [CanIfRxPduUserRxIndicationUL](#), [CanIfRxPduUserRxIndicationName](#)) to be called with its providing receive indication service for the received L-SDU.] ()

**[SWS\_CANIF\_00135]** [If a target upper layer module was configured to be called with its providing receive indication service (see [SWS\_CANIF\_00056]), the CanIf shall call this configured receive indication callback service (see [CanIfRxPduUserRxIndicationName](#)) and shall provide the parameters required for upper layer notification

callback functions (see [SWS\_CANIF\_00012]) based on the parameters of `CanIf_RxIndication()`.] (SRS\_BSW\_00325)

Note: A single receive L-PDU can only be assigned to a single receive indication callback service (refer to multiplicity of `CanIfRxPduUserRxIndicationName`).

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

- 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.

The function `CanIf_ReadRxPduData()` makes reading out data without dependence of reception event (`RxIndication`) possible. When it is enabled at configuration time (see `CanIfPublicReadRxPduDataApi`), not necessarily a receive indication service for the same L-SDU has to be configured (see `CanIfRxPduUserRxIndicationUL`). If needed, the receive indication can be enabled, too.

By this way the type of mechanism to receive L-SDUs (in the upper layer modules of `CanIf`) can be chosen at configuration time by the parameter `CanIfRxPduUserRxIndicationUL` and parameter `CanIfRxPduReadData` according to the needs of the upper layer module, to which the corresponding receive L-SDU belongs to. For details please refer to section 9.9 “Read received data”.

**[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 pre-emptive 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** [`CanIf` shall protect preemptive events, which access shared resources, that could be changed during `CanIf`'s event handling, against each other.](*SRS\_BSW\_00312*)

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 `Can-Drvs`.

**[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`.] ([SRS\\_Can\\_01162](#), [SRS\\_Can\\_02003](#))

**[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 corresponding upper layer modules about failed transmission by calling `<User_TxConfirmation>(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 `<User_TxConfirmation>()` 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 subsection 8.6.3.9 “<User\_ControllerModeIndication>”.

**[SWS\_CANIF\_00711]** [When callback `CanIf_ControllerModeIndication(ControllerId, ControllerMode)` is called, `CanIf` shall call `CanSM_ControllerModeIndication(ControllerId, ControllerMode)` of the `CanSm` or a `CDD` (see [\[SWS\\_CANIF\\_00691\]](#), [\[SWS\\_CANIF\\_00692\]](#)).] ()

Note for [\[SWS\\_CANIF\\_00711\]](#): See subsection 8.6.3.9 “<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 subsection 8.6.3.9 “<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 [13].

### 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 [\[13\]](#)).

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 [\[13\]](#)).

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`.

](*SRS\_Can\_01151*)

**[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`.](*SRS\_Can\_01136*)

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

- `sources: EcuM_WakeupSourceType` (see [13])

**[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

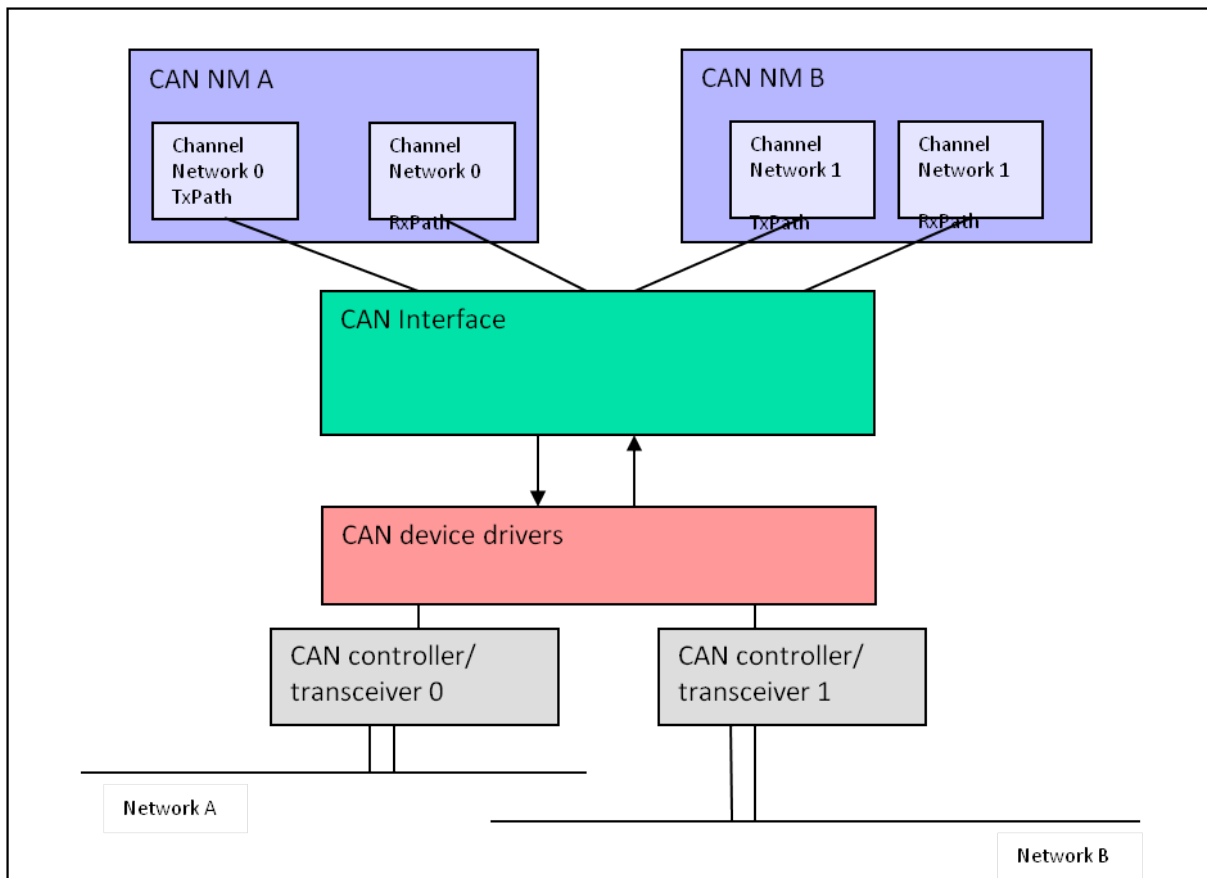
### 7.19.1 PDU channel groups

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

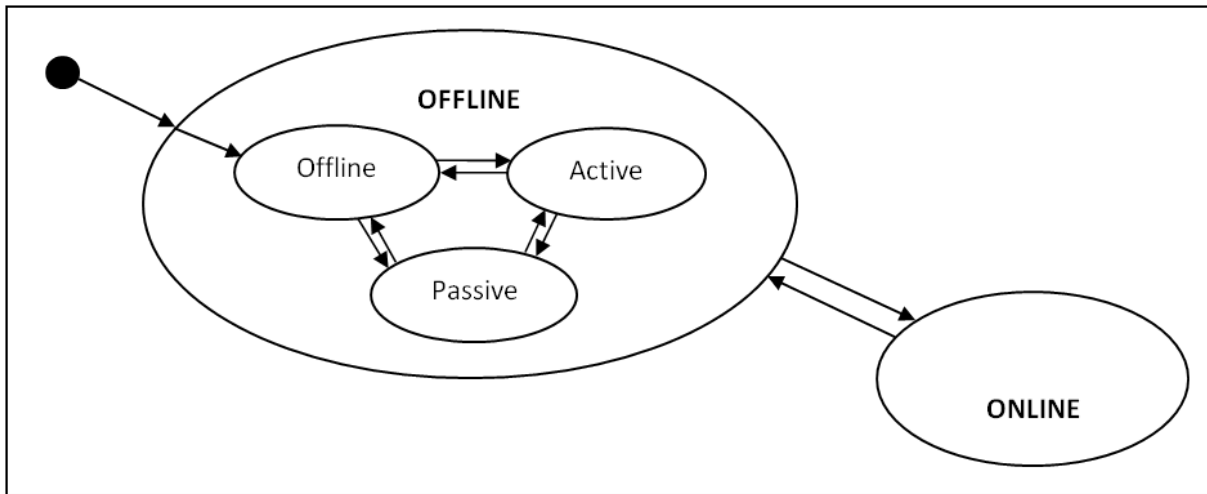
CanIf provides the services `CanIf_SetPduMode()` and `CanIf_GetPduMode()` to prevent the processing of

- 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 communication the PDU channel modes `CANIF_TX_OFFLINE` and `CANIF_TX_OFFLINE_ACTIVE` enable/disable transmission path separately.

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(ControllorId, 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<sub>N</sub> 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.](SRS\_Can\_02003)

**[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 `CanIfRxPduXLParams`, it matches any received value.]([SRS\\_Can\\_02003](#))

**[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.]([SRS\\_Can\\_02003](#))

**[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.]([SRS\\_Can\\_02003](#))

**[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.]([SRS\\_Can\\_02003](#))

**[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.]([SRS\\_Can\\_02003](#))

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.

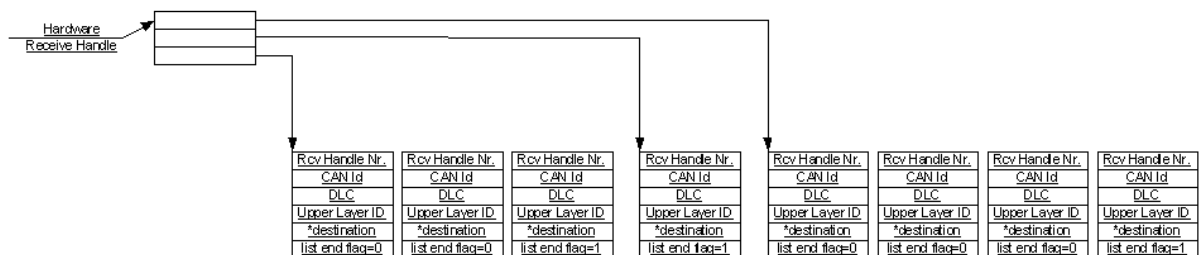
**[SWS\_CANIF\_00645]** [A range of **CanIds** which shall pass the software receive filter shall either be defined by its upper limit (see **CanIfHrhRangeRxPduUpperCanId**) and lower limit (see **CanIfHrhRangeRxPduLowerCanId**) **CanId**, or by a base ID (see **CanIfHrhRangeBaseId**) and a mask that 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**.] (**SRS\_Can\_01018**)



**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 then the configured Data Length value (see `CanIfRxPduDataLength`).] (*SRS\_Can\_01005*)

**[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 the target upper layer module (see [SWS\_CANIF\_00135]), if the Data Length Check is passed.] ()

**[SWS\_CANIF\_00830]** [`CanIf` shall pass the received (see [SWS\_CANIF\_00006]) length value to the target upper layer module (see [SWS\_CANIF\_00135]), if the Data Length Check is not configured (see `CanIfPrivateDataLengthCheck` and `CanIfRxPduDataLengthCheck`)] ()

## 7.22 L-SDU dispatcher to upper layers

Rationale: At transmission side the **L-SDU** dispatcher has to find out the corresponding Tx confirmation callback service of the target upper layer module. At reception side each **L-SDU** belongs to one single upper layer module as destination. This relation is assigned statically at configuration time. The task of the **L-SDU** dispatcher inside of **CanIf** is to find out the customer for a received **L-SDU** and to dispatch the indications towards the found upper layer. These transmit confirmation as well as receive indication notification services may exist several times with different names defined in the notified upper layer modules. Those notification services are statically configured, depending on the layers that have to be served.

## 7.23 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 [section 8.4 "Callback notifications"](#)).

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 [section 7.11 "Transmit buffering"](#)).

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

## 7.24 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.

### 7.24.1 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.

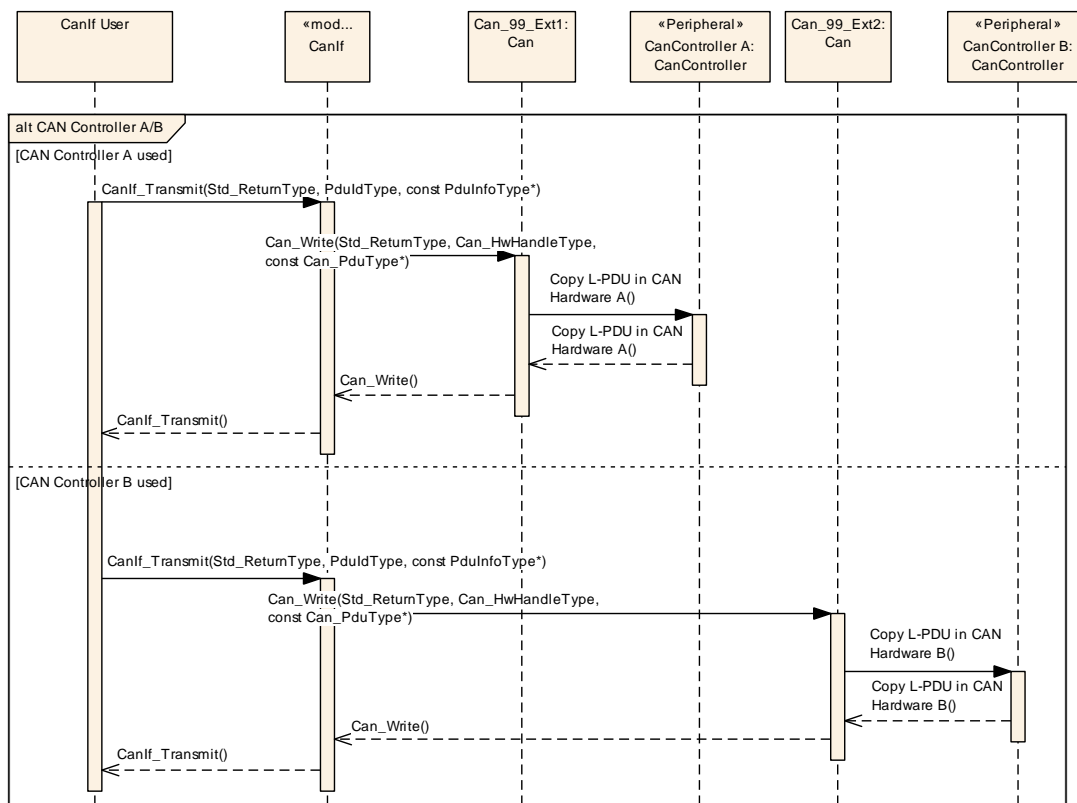
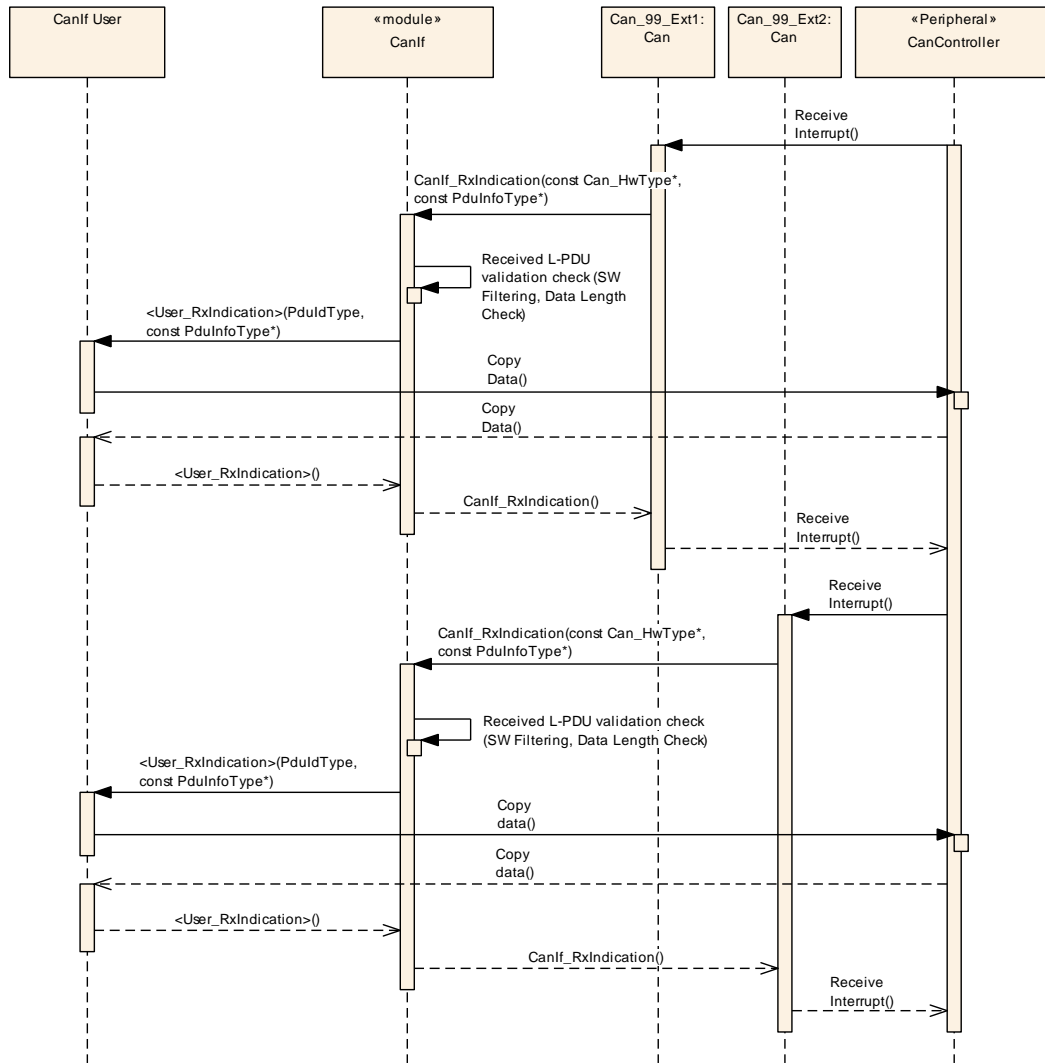


Figure 7.10: Transmission request with multiple CAN Drivers - simplified

Operations called	Description
CanIf_Transmit (PduId_1, PduInfoPtr_1)	Upper layer initiates a <i>transmit request</i> . 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.

## 7.24.2 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 CanDvs - simplified**

Operations called	Description
Receive Interrupt	CAN Controller 1 signals a successful reception and triggers a <i>receive interrupt</i> . The <i>ISR</i> 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.
<User_RxIndication> (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 <b>CAN Controller 2</b> signals a successful reception and triggers a <i>receive interrupt</i> . The <i>ISR</i> of <b>CanDrv B</b> is invoked.
<b>CanIf_RxIndication</b> ( <b>Mailbox_2</b> , <b>PduInfoPtr_2</b> )	The reception is indicated to <b>CanIf</b> by calling of <b>CanIf_RxIndication()</b> . The pointer <b>Mailbox_2</b> identifies the <b>HRH</b> and its corresponding <b>CAN Controller</b> , which contains the received <b>L-PDU</b> specified by <b>PduInfoPtr_2</b> .
Validation check (SW Filtering, Data Length Check)	The Software Filtering checks, whether the <b>Received L-PDU</b> will be processed on a local ECU. If not, the <b>Received L-SDU</b> is not indicated to upper layers and further processing is suppressed. If the <b>L-PDU</b> is found, the Data Length of the <b>Received L-PDU</b> is compared with the expected, statically configured one for the received <b>L-PDU</b> .
< <b>User_RxIndication</b> > ( <b>CanRxPduId_2</b> , <b>CanPduInfoPtr_2</b> )	The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter <b>CanRxPduId_2</b> specifies the ID of the received <b>L-SDU</b> . The second parameter is the reference on <b>PduInfoType</b> which provides access to the buffer containing the <b>L-SDU</b> .

## 7.25 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*.] (**SRS\_Can\_01174**)

**[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** (see **CanIfTxPduPnFilterPdu**).] (**SRS\_Can\_01174**)

**[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**.] (**SRS\_Can\_01173**, **SRS\_Can\_01174**)

**[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 [\[SWS\\_CANIF\\_00747\]](#)).]()

## 7.26 CAN FD Support

For performance reasons some `CAN Controllers` allow to use a Flexible Data-Rate feature called `CAN FD` (see [\[12, ISO 11898-1:2015\]](#)). 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 configured `CanIfTxPduCanIdType`. `CanIf` indicates this to `CanDrv` utilizing the second most significant 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`  $\leq$  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 (see `CanIfRxPduCanIdType`). 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.



## 7.27 CAN XL Support

CAN XL (see [14, 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 [14, CiA610-1] and [15, 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 [15, 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 addressing 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 capabilities.
- `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 `CanControllerXLBaudrateConfig`).

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 [section 7.11](#)) is not available for CAN XL frames.

### 7.27.1 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 `CanIfRxPdu`, 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 `CanIfTxPdu`.

**[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.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

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 [section 10.2](#).

## 7.28 Security Events

**[SWS\_CANIF\_91010]** Security events for `CanIf` [

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

] ([RS\\_Ids\\_00810](#))

**[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 AUTOSAR\_SWS\_BSWGeneral. [\]\(RS\\_Ids\\_00810\)](#)

**[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 `CanIf` shall report the security event `CANIF_SEV_TX_ERROR_DETECTED`.

The context data is structured as follows:

Context Data (2 Byte)

- ControllerID (1 Byte)
- CanError (1 Byte)
  - CAN\_ERROR\_BIT\_MONITORING1 (0x1)
  - CAN\_ERROR\_BIT\_MONITORING0 (0x2)
  - CAN\_ERROR\_BIT (0x3)
  - CAN\_ERROR\_CHECK\_ACK\_FAILED (0x4)
  - CAN\_ERROR\_ACK\_DELIMITER (0x5)
  - CAN\_ERROR\_ARBITRATION\_LOST (0x6)
  - CAN\_ERROR\_OVERLOAD (0x7)

[\]\(RS\\_Ids\\_00810\)](#)

**[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 `CANIF_SEV_RX_ERROR_DETECTED`.

The context data is structured as follows:

Context Data (2 Byte)

- ControllerID (1 Byte)
- CanError (1 Byte)
  - CAN\_ERROR\_CHECK\_FORM\_FAILED (0x8)
  - CAN\_ERROR\_CHECK\_STUFFING\_FAILED (0x9)
  - CAN\_ERROR\_CHECK\_CRC\_FAILED (0xA)
  - CAN\_ERROR\_BUS\_LOCK (0xB)

[\]\(RS\\_Ids\\_00810\)](#)

**[SWS\_CANIF\_00917]** [If `CanIf_ControllerErrorStatePassive()` is called by `CanDrv`, the `CanIf` shall report the security event `CANIF_SEV_ERRORSTATE_PASSIVE` in following cases:

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

The context data is structured as follows:

Context Data (2 Byte)

- ControllerID (1 Byte)
- ErrorCounterThreshold (1 Byte)
  - TxErrorCounter > 127 AND RxErrorCounter > 127(0x0)
  - TxErrorCounter > 127 AND RxErrorCounter < 127 (0x1)
  - RxErrorCounter > 127 AND TxErrorCounter < 127 (0x2)

](RS\_Ids\_00810)

**[SWS\_CANIF\_00918]** [If CanIf\_ControllerBusOff is called by CanDrv, the CanIf shall report the security event CANIF\_SEV\_ERRORSTATE\_BUSOFF. The context data is structured as follows:

Context Data (1 Byte)

- Controller ID (1 Byte)

](RS\_Ids\_00810)

## 7.29 Error classification

### 7.29.1 Development Errors

**[SWS\_CANIF\_91006]** [

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

](0)

## 7.29.2 Runtime Errors

[SWS\_CANIF\_91007] [

<i>Type of error</i>	<i>Related error code</i>	<i>Error value</i>
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 Transient Faults

There are no transient faults.

## 7.29.4 Production Errors

There are no production errors.

## 7.29.5 Extended Production Errors

There are no extended production errors.

## 8 API specification

### 8.1 Imported types

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

[SWS\_CANIF\_00142] [

<i>Module</i>	<i>Header File</i>	<i>Imported Type</i>
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
ComStack_Types	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
EcuM	EcuM.h	EcuM_WakeupSourceType
IdsM	IdsM_Types.h	IdsM_SecurityEventIdType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

]([SRS\\_BSW\\_00348](#), [SRS\\_BSW\\_00353](#), [SRS\\_BSW\\_00361](#))

### 8.2 Type definitions

#### 8.2.1 CanIf\_ConfigType

[SWS\_CANIF\_00144] [

<b>Name</b>	CanIf_ConfigType	
<b>Kind</b>	Structure	
<b>Elements</b>	implementation specific	
	<b>Type</b>	-



△

	<b>Comment</b>	The contents of the initialization data structure are CAN interface specific
<b>Description</b>	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.	
<b>Available via</b>	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]** [

<b>Name</b>	CanIf_PduModeType		
<b>Kind</b>	Enumeration		
<b>Range</b>	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 <code>CanIfTxOfflineActiveSupport = TRUE</code> .
	CANIF_ONLINE	0x03	Transmit and receive path of the corresponding channel are enabled => full operation mode
<b>Description</b>	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.		
<b>Available via</b>	CanIf.h		

]()

### 8.2.3 CanIf\_NotifStatusType

[SWS\_CANIF\_00201] [

<b>Name</b>	CanIf_NotifStatusType		
<b>Kind</b>	Enumeration		
<b>Range</b>	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.
<b>Description</b>	Return value of CAN L-PDU notification status.		
<b>Available via</b>	CanIf.h		

]()

## 8.3 Function definitions

[SWS\_CANIF\_00661] [All CanIf API services other than [CanIf\\_Init\(\)](#) and [CanIf\\_GetVersionInfo\(\)](#) 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] [

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

] ([SRS\\_BSW\\_00405](#), [SRS\\_BSW\\_00101](#), [SRS\\_BSW\\_00358](#), [SRS\\_BSW\\_00414](#), [SRS\\_Can\\_01021](#), [SRS\\_Can\\_01022](#))

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\_Delnit

[SWS\_CANIF\_91002] [

<b>Service Name</b>	CanIf_Delnit
<b>Syntax</b>	<code>void CanIf_DeInit (</code> <code>    void</code> <code>)</code>
<b>Service ID [hex]</b>	0x02
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	De-initializes the CanIf module.
<b>Available via</b>	CanIf.h

] ([SRS\\_Can\\_01168](#), [SRS\\_BSW\\_00336](#))

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] [

<b>Service Name</b>	CanIf_SetControllerMode	
<b>Syntax</b>	<code>Std_ReturnType CanIf_SetControllerMode (</code> <code>    uint8 ControllerId,</code> <code>    Can_ControllerStateType ControllerMode</code> <code>)</code>	
<b>Service ID [hex]</b>	0x03	
<b>Sync/Async</b>	Asynchronous	
<b>Reentrancy</b>	Reentrant (Not for the same controller)	
<b>Parameters (in)</b>	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for mode transition.
	ControllerMode	Requested mode transition
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	





<b>Return value</b>	Std_ReturnType	E_OK: Controller mode request has been accepted E_NOT_OK: Controller mode request has not been accepted
<b>Description</b>	This service calls the corresponding CAN Driver service for changing of the CAN controller mode.	
<b>Available via</b>	CanIf.h	

]([SRS\\_Can\\_01027](#))

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]** [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.]([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00774]** [If parameter `ControllerMode` of `CanIf_SetControllerMode()` has an invalid value (not `CAN_CS_STARTED`, `CAN_CS_SLEEP` or `CAN_CS_STOPPED`), the `CanIf` shall report development error code `CANIF_E_PARAM_CTRLMODE` to the `Det_ReportError` service of the DET module, when `CanIf_SetControllerMode()` is called.]([SRS\\_BSW\\_00323](#))

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

### 8.3.4 CanIf\_GetControllerMode

**[SWS\_CANIF\_00229]** [

<b>Service Name</b>	CanIf_GetControllerMode	
<b>Syntax</b>	Std_ReturnType CanIf_GetControllerMode ( uint8 ControllerId, Can_ControllerStateType* ControllerModePtr )	
<b>Service ID [hex]</b>	0x04	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for current operation mode.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	ControllerModePtr	Pointer to a memory location, where the current mode of the CAN controller will be stored.





<b>Return value</b>	Std_ReturnType	E_OK: Controller mode request has been accepted. E_NOT_OK: Controller mode request has not been accepted.
<b>Description</b>	This service calls the corresponding CAN Driver service for obtaining the current status of the CAN controller.	
<b>Available via</b>	CanIf.h	

]([SRS\\_Can\\_01028](#))

**[SWS\_CANIF\_00313]** [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.]([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00656]** [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.]([SRS\\_BSW\\_00323](#))

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]** [

<b>Service Name</b>	CanIf_GetControllerErrorState	
<b>Syntax</b>	Std_ReturnType CanIf_GetControllerErrorState ( uint8 ControllerId, Can_ErrorStateType* ErrorStatePtr )	
<b>Service ID [hex]</b>	0x4b	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant for the same ControllerId	
<b>Parameters (in)</b>	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	ErrorStatePtr	Pointer to a memory location, where the error state of the CAN controller will be stored.
<b>Return value</b>	Std_ReturnType	E_OK: Error state request has been accepted. E_NOT_OK: Error state request has not been accepted.
<b>Description</b>	This service calls the corresponding CAN Driver service for obtaining the error state of the CAN controller.	
<b>Available via</b>	CanIf.h	

]([SRS\\_Can\\_01169](#))

**[SWS\_CANIF\_00898]** [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.]([SRS\\_BSW\\_00323](#))

[SWS\_CANIF\_00899] [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.] (SRS\_BSW\_00323)

### 8.3.6 CanIf\_Transmit

[SWS\_CANIF\_00005] [

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

] (SRS\_Can\_01008)

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.] (SRS\_Can\_01162, SRS\_Can\_02003)

[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\_00243]** [`CanIf` shall set the two most significant bits ('Identifier Extension flag' (see [12, ISO 11898-1:2015]) and 'CAN FD flag') of the *CanId* (`PduInfoPtr->id`) before `CanIf` passes the predefined *CanId* to `CanDrv` at call of `Can_Write()` (see [1, Specification of CAN Driver], definition of `Can_IdType` [SWS\_Can\_00416]). The *CanId* format type of each CAN L-PDU can be configured by `CanIfTxPduCanIdType`, refer to `CanIfTxPduCanIdType`.] ([SRS\\_Can\\_01141](#))

**[SWS\_CANIF\_00957]** [`CanIf_Transmit()` shall call `CanXL_Write()` with the hardware transmit handle corresponding to the provided `TxPduId` 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)

] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_01162](#), [SRS\\_Can\\_02003](#))

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]** [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.] ([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00320]** [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.]([SRS\\_BSW\\_00323](#))

**[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*.]([SRS\\_Can\\_01162](#), [SRS\\_Can\\_02003](#))

**[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*.]([SRS\\_Can\\_02003](#))

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 [\[ECUC\\_EcuC\\_00078\]](#)) referenced by `TxPduId` and `CanIfTxPduTruncation` is enabled, `CanIf` shall transmit data up to the length of the global PDU (see [\[ECUC\\_EcuC\\_00078\]](#)) and discard the rest.](/)

**[SWS\_CANIF\_00900]** [When `CanIf_Transmit()` is called with `PduInfoPtr->SduLength` exceeding the length of the global PDU (see [\[ECUC\\_EcuC\\_00078\]](#)) 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 `->MetaDataPtr`.] ([SRS\\_Can\\_02003](#))

As mentioned in [subsection 7.27.1](#), 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 be transmitted in the `Acceptance Field`. The `CAN ID` is either configured with the parameters `CanIfTxPduCanId` and `CanIfTxPduCanIdType` or passed as `MetaData`.] ([SRS\\_Can\\_02003](#))

[SWS\_CANIF\_00943] [When `CanIf_Transmit` is called and `CanXL_Write` is used for transmission the parameter `PduInfo->sec` shall be set to 0.] ([SRS\\_Can\\_02003](#))

### 8.3.7 CanIf\_ReadRxPduData

[SWS\_CANIF\_00194] [

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

] ([SRS\\_Can\\_01125](#), [SRS\\_Can\\_01129](#))

[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] [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_RX-PDUID` to the `Det_ReportError` service of the DET, when `CanIf_ReadRxPduData()` is called.] ([SRS\\_BSW\\_00323](#))



**[SWS\_CANIF\_00326]** [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.] ([SRS\\_BSW\\_00323](#))

**[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`.] ()

### 8.3.8 CanIf\_ReadTxNotifStatus

**[SWS\_CANIF\_00202]** [

<b>Service Name</b>	CanIf_ReadTxNotifStatus	
<b>Syntax</b>	<code>CanIf_NotifStatusType CanIf_ReadTxNotifStatus ( PduIdType CanIfTxSduId )</code>	
<b>Service ID [hex]</b>	0x07	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	<code>CanIfTxSduId</code>	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	<code>CanIf_NotifStatusType</code>	Current confirmation status of the corresponding CAN Tx L-PDU.
<b>Description</b>	This service returns the confirmation status (confirmation occurred or not) of a specific static or dynamic CAN Tx L-PDU, requested by the <code>CanIfTxSduId</code> .	
<b>Available via</b>	CanIf.h	

] ([SRS\\_Can\\_01130](#))

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]** [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. [\(SRS\\_BSW\\_00323\)](#)

**[SWS\_CANIF\_00335]** [Configuration of `CanIf_ReadTxNotifStatus()`: This API can be enabled or disabled at pre-compile time configuration globally by the parameter `CanIfPublicReadTxPduNotifStatusApi()`.]

### 8.3.9 CanIf\_ReadRxNotifStatus

**[SWS\_CANIF\_00230]** [

<b>Service Name</b>	CanIf_ReadRxNotifStatus	
<b>Syntax</b>	<code>CanIf_NotifStatusType CanIf_ReadRxNotifStatus ( PduIdType CanIfRxSduId )</code>	
<b>Service ID [hex]</b>	0x08	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	CanIfRxSdul	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	<code>CanIf_NotifStatusType</code>	Current indication status of the corresponding CAN Rx L-PDU.
<b>Description</b>	This service returns the indication status (indication occurred or not) of a specific CAN Rx L-PDU, requested by the <code>CanIfRxSdul</code> .	
<b>Available via</b>	CanIf.h	

]([SRS\\_Can\\_01130](#), [SRS\\_Can\\_01131](#))

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

**[SWS\_CANIF\_00394]** [If configuration parameters `CanIfPublicReadRxPduNotifStatusApi` and `CanIfRxPduReadNotifStatus` 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]** [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. [\(SRS\\_BSW\\_00323\)](#)

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.`]

### 8.3.10 CanIf\_SetPduMode

**[SWS\_CANIF\_00008]** [

<b>Service Name</b>	CanIf_SetPduMode	
<b>Syntax</b>	<pre>Std_ReturnType CanIf_SetPduMode (     uint8 ControllerId,     CanIf_PduModeType PduModeRequest )</pre>	
<b>Service ID [hex]</b>	0x09	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	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
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: Request for mode transition has been accepted. E_NOT_OK: Request for mode transition has not been accepted.
<b>Description</b>	This service sets the requested mode at the L-PDUs of a predefined logical PDU channel.	
<b>Available via</b>	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]** [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.]([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00860]** [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.]([SRS\\_BSW\\_00323](#))

**[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`.]()

### 8.3.11 CanIf\_GetPduMode

[SWS\_CANIF\_00009] [

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

]()

[SWS\_CANIF\_00346] [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.]([SRS\\_BSW\\_00323](#))

[SWS\_CANIF\_00657] [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.]([SRS\\_BSW\\_00323](#))

### 8.3.12 CanIf\_GetVersionInfo

[SWS\_CANIF\_00158] [

<b>Service Name</b>	CanIf_GetVersionInfo	
<b>Syntax</b>	<pre>void CanIf_GetVersionInfo (     Std_VersionInfoType* VersionInfo )</pre>	
<b>Service ID [hex]</b>	0x0b	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	VersionInfo	Pointer to where to store the version information of this module.
<b>Return value</b>	None	
<b>Description</b>	This service returns the version information of the called CAN Interface module.	
<b>Available via</b>	CanIf.h	

]([SRS\\_BSW\\_00407](#), [SRS\\_BSW\\_00411](#))

### 8.3.13 CanIf\_SetDynamicTxId

[SWS\_CANIF\_00189] [

<b>Service Name</b>	CanIf_SetDynamicTxId	
<b>Syntax</b>	<pre>void CanIf_SetDynamicTxId (     PduIdType CanIfTxSduId,     Can_IdType CanId )</pre>	
<b>Service ID [hex]</b>	0x0c	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service reconfigures the corresponding CAN identifier of the requested CAN L-PDU.	
<b>Available via</b>	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.]([SRS\\_Can\\_02003](#))

[SWS\_CANIF\_00352] [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.]([SRS\\_BSW\\_00323](#))

[SWS\_CANIF\_00353] [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.]([SRS\\_BSW\\_00323](#))

[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] [

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

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

[SWS\_CANIF\_00538] [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.](SRS\_BSW\_00323)

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]** [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.]([SRS\\_BSW\\_00323](#))

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]** [

<b>Service Name</b>	<code>CanIf_GetTrcvMode</code>	
<b>Syntax</b>	<pre>Std_ReturnType CanIf_GetTrcvMode (     uint8 TransceiverId,     CanTrcv_TrcvModeType* TransceiverModePtr )</pre>	
<b>Service ID [hex]</b>	0x0e	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	<code>TransceiverId</code>	Abstracted <code>CanIf</code> <code>TransceiverId</code> , which is assigned to a CAN transceiver, which is requested for current operation mode.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	<code>TransceiverModePtr</code>	Requested mode of requested network the Transceiver is connected to.
<b>Return value</b>	<code>Std_ReturnType</code>	<code>E_OK</code> : Transceiver mode request has been accepted. <code>E_NOT_OK</code> : Transceiver mode request has not been accepted.
<b>Description</b>	This function invokes <code>CanTrcv_GetOpMode</code> and updates the parameter <code>TransceiverModePtr</code> with the value <code>OpMode</code> provided by <code>CanTrcv</code> .	
<b>Available via</b>	<code>CanIf.h</code>	

]()

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]** [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.] ([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00650]** [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.] ([SRS\\_BSW\\_00323](#))

**[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]** [

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

] ()



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

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

**[SWS\_CANIF\_00537]** [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.](*SRS\_BSW\_00323*)

**[SWS\_CANIF\_00649]** [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.](*SRS\_BSW\_00323*)

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 unpowered. Then one may be able to return `CANIF_TRCV_WU_POWER_ON`, whereas the other may state e.g. `CANIF_TRCV_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] [

<b>Service Name</b>	CanIf_SetTrcvWakeupMode	
<b>Syntax</b>	Std_ReturnType CanIf_SetTrcvWakeupMode ( uint8 TransceiverId, CanTrcv_TrvcWakeupModeType TrcvWakeupMode )	
<b>Service ID [hex]</b>	0x10	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	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
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	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.
<b>Description</b>	This function shall call CanTrcv_SetTrcvWakeupMode.	
<b>Available via</b>	CanIf.h	

](SRS\_Can\_01175)

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\\_TrvcWakeupModeType](#) (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`.

CANIF\_TRCV\_WU\_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\\_SetTrcvWakeupMode\(\)](#) (depending on the implementation).

CANIF\_TRCV\_WU\_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.

CANIF\_TRCV\_WU\_CLEAR:

If notification of wake-up events is disabled (see description of mode `CANIF_TRCV_WU_DISABLE`), detected wake-up events are buffered. Calling `CanIf_SetTrcvWakeupMode()` with parameter `CANIF_TRCV_WU_CLEAR` clears these buffered 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]** [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.] ([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00536]** [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.] ([SRS\\_BSW\\_00323](#))

**[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]** [

<b>Service Name</b>	CanIf_CheckWakeup	
<b>Syntax</b>	Std_ReturnType CanIf_CheckWakeup ( EcuM_WakeupSourceType WakeupSource )	
<b>Service ID [hex]</b>	0x11	
<b>Sync/Async</b>	Asynchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	WakeupSource	Source device, which initiated the wake up event: CAN controller or CAN transceiver
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	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





<b>Description</b>	This service checks, whether an underlying CAN driver or a CAN transceiver driver already signals a wakeup event.
<b>Available via</b>	CanIf.h

]()

Note: *Integration Code* calls this function

**[SWS\_CANIF\_00398]** [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.] ([SRS\\_BSW\\_00323](#))

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]** [

<b>Service Name</b>	CanIf_CheckValidation	
<b>Syntax</b>	Std_ReturnType CanIf_CheckValidation ( EcuM_WakeupSourceType WakeupSource )	
<b>Service ID [hex]</b>	0x12	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	WakeupSource	Source device which initiated the wake-up event and which has to be validated: CAN controller or CAN transceiver
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	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.
<b>Description</b>	This service is performed to validate a previous wakeup event.	
<b>Available via</b>	CanIf.h	

]()

Note: *Integration Code* calls this function

**[SWS\_CANIF\_00404]** [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.] ([SRS\\_BSW\\_00323](#))

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]** [

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

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

[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`.]()

### 8.3.22 CanIf\_CheckTrcvWakeFlag

[SWS\_CANIF\_00761] [

<b>Service Name</b>	CanIf_CheckTrcvWakeFlag	
<b>Syntax</b>	Std_ReturnType CanIf_CheckTrcvWakeFlag ( uint8 TransceiverId )	
<b>Service ID [hex]</b>	0x1f	
<b>Sync/Async</b>	Asynchronous	
<b>Reentrancy</b>	Reentrant for different CAN transceivers	
<b>Parameters (in)</b>	TransceiverId	Abstract CanIf TransceiverId, which is assigned to the designated CAN transceiver.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	





<b>Return value</b>	Std_ReturnType	E_OK: Request has been accepted E_NOT_OK: Request has not been accepted
<b>Description</b>	Requests the CanIf module to check the Wake flag of the designated CAN transceiver.	
<b>Available via</b>	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 called.]()

**[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`.]()

### 8.3.23 CanIf\_SetBaudrate

**[SWS\_CANIF\_00867]** [

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

]()

**[SWS\_CANIF\_00868]** [The service `CanIf_SetBaudrate()` shall call `Can_SetBaudrate(Controller, BaudRateConfigID)` for the requested `CAN Controller`.]()

**[SWS\_CANIF\_00869]** [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.]([SRS\\_BSW\\_00323](#))

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]** [

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

]()

**[SWS\_CANIF\_00907]** [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.]([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00908]** [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.]([SRS\\_BSW\\_00323](#))

### 8.3.25 CanIf\_GetControllerTxErrorCounter

[SWS\_CANIF\_91004] [

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

]()

[SWS\_CANIF\_00909] [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.] ([SRS\\_BSW\\_00323](#))

[SWS\_CANIF\_00910] [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.] ([SRS\\_BSW\\_00323](#))

### 8.3.26 CanIf\_EnableBusMirroring

[SWS\_CANIF\_91005] [

<b>Service Name</b>	CanIf_EnableBusMirroring	
<b>Syntax</b>	<pre>Std_ReturnType CanIf_EnableBusMirroring (     uint8 ControllerId,     boolean MirroringActive )</pre>	
<b>Service ID [hex]</b>	0x4c	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	







<b>Parameters (in)</b>	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: Mirroring mode was changed. E_NOT_OK: Wrong ControllerId, or mirroring globally disabled (see CanIfBusMirroringSupport).
<b>Description</b>	Enables or disables mirroring for a CAN controller.	
<b>Available via</b>	CanIf.h	

](0)

**[SWS\_CANIF\_00911]** [If Bus Mirroring is not enabled (see [CanIfBusMirroringSupport](#)), the API `CanIf_EnableBusMirroring()` can be omitted.]([SRS\\_Can\\_01172](#))

**[SWS\_CANIF\_00912]** [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.]([SRS\\_BSW\\_00323](#))

### 8.3.27 CanIf\_GetCurrentTime

**[SWS\_CANIF\_91014]{DRAFT}** [

<b>Service Name</b>	CanIf_GetCurrentTime (draft)	
<b>Syntax</b>	Std_ReturnType CanIf_GetCurrentTime ( uint8 Controller, Can_TimeStampType* timeStampPtr )	
<b>Service ID [hex]</b>	0x51	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	Controller	Index of the addresses CAN controller.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	timeStampPtr	current time stamp
<b>Return value</b>	Std_ReturnType	E_OK: successful E_NOT_OK: failed
<b>Description</b>	This service calls the corresponding CAN Driver service to retrieve the current time value out of the HW registers. <b>Tags:</b> atp.Status=draft	
<b>Available via</b>	CanIf.h	

]([SRS\\_Can\\_01181](#))

**[SWS\_CANIF\_00922]{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]{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]{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]{DRAFT}** [The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`.]()

### 8.3.28 CanIf\_EnableEgressTimeStamp

**[SWS\_CANIF\_91011]{DRAFT}** [

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

] ([SRS\\_Can\\_01181](#))

**[SWS\_CANIF\_00926]{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]{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]{DRAFT}** [The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`.]()

### 8.3.29 CanIf\_GetEgressTimeStamp

[SWS\_CANIF\_91012]{DRAFT} [

<b>Service Name</b>	CanIf_GetEgressTimeStamp (draft)	
<b>Syntax</b>	<pre>Std_ReturnType CanIf_GetEgressTimeStamp (     PduIdType TxPduId,     Can_TimeStampType* timeStampPtr )</pre>	
<b>Service ID [hex]</b>	0x53	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant for the same TxPduId, Reentrant for different TxPduId	
<b>Parameters (in)</b>	TxPduId	L-PDU handle of CAN L-PDU for which the time stamp shall be returned.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	timeStampPtr	current time stamp
<b>Return value</b>	Std_ReturnType	E_OK: successful E_NOT_OK: failed
<b>Description</b>	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. <b>Tags:</b> atp.Status=draft	
<b>Available via</b>	CanIf.h	

] ([SRS\\_Can\\_01181](#))

[SWS\_CANIF\_00929]{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]{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]{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]{DRAFT} [The function shall be pre compile time configurable On/Off by the configuration parameter: [CanIfGlobalTimeSupport](#).]()

### 8.3.30 CanIf\_GetIngressTimeStamp

[SWS\_CANIF\_91013]{DRAFT} [

<b>Service Name</b>	CanIf_GetIngressTimeStamp (draft)	
<b>Syntax</b>	<pre>Std_ReturnType CanIf_GetIngressTimeStamp (     PduIdType RxPduId,     Can_TimeStampType* timeStampPtr )</pre>	





<b>Service ID [hex]</b>	0x54	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant for the same RxPduId, Reentrant for different RxPduIds	
<b>Parameters (in)</b>	RxPduId	ID of the received I-PDU for which the time stamp shall be returned.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	timeStampPtr	current time stamp
<b>Return value</b>	Std_ReturnType	E_OK: successful E_NOT_OK: failed
<b>Description</b>	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. <b>Tags:</b> atp.Status=draft	
<b>Available via</b>	CanIf.h	

](SRS\_Can\_01181)

**[SWS\_CANIF\_00933]{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]{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]{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]{DRAFT}** [The function shall be pre compile time configurable On/Off by the configuration parameter: `CanIfGlobalTimeSupport`.]()

## 8.4 Callback notifications

This is a list of functions provided for other modules.

### 8.4.1 CanIf\_TriggerTransmit

**[SWS\_CANIF\_00883]** [

<b>Service Name</b>	CanIf_TriggerTransmit
<b>Syntax</b>	<pre>Std_ReturnType CanIf_TriggerTransmit (     PduIdType TxPduId,     PduInfoType* PduInfoPtr )</pre>
<b>Service ID [hex]</b>	0x41





<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant for different PduIds. Non reentrant for the same PduId.	
<b>Parameters (in)</b>	TxPduId	ID of the SDU that is requested to be transmitted.
<b>Parameters (inout)</b>	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLength. On return, the service will indicate the length of the copied SDU data in SduLength.
<b>Parameters (out)</b>	None	
<b>Return value</b>	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.
<b>Description</b>	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.	
<b>Available via</b>	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 corresponding `<User_TriggerTransmit>()` function, passing the translated `TxPduId` and the pointer to the `PduInfo` structure (`PduInfoPtr`). Upon return, `CanIf_TriggerTransmit()` shall return the return value of its `<User_TriggerTransmit>()`.]()

## 8.4.2 CanIf\_TxConfirmation

**[SWS\_CANIF\_00007]** [

<b>Service Name</b>	CanIf_TxConfirmation	
<b>Syntax</b>	<pre>void CanIf_TxConfirmation (     PduIdType CanTxPduId )</pre>	
<b>Service ID [hex]</b>	0x13	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service confirms a previously successfully processed transmission of a CAN TxPDU.	
<b>Available via</b>	CanIf_Can.h	

](SRS\_Can\_01009)

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 [16, 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]** [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.](SRS\_BSW\_00323)

**[SWS\_CANIF\_00412]** [If `CanIf` was not initialized before calling `CanIf_TxConfirmation()`, `CanIf` shall not call the service `<User_TxConfirmation>()` 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).

**[SWS\_CANIF\_00414]** [Configuration of `CanIf_TxConfirmation()`: Each Tx L-PDU (see `CanIfTxPduCfg`) has to be configured with a corresponding transmit confirmation service of an upper layer module (see [SWS\_CANIF\_00011]) which is called in `CanIf_TxConfirmation()`.]()

### 8.4.3 CanIf\_RxIndication

**[SWS\_CANIF\_00006]** [

<b>Service Name</b>	CanIf_RxIndication	
<b>Syntax</b>	<pre>void CanIf_RxIndication (     const Can_HwType* Mailbox,     const PduInfoType* PduInfoPtr )</pre>	
<b>Service ID [hex]</b>	0x14	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Mailbox	Identifies the HRH and its corresponding CAN Controller
	PduInfoPtr	Pointer to the received L-PDU
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	





<b>Return value</b>	None
<b>Description</b>	This service indicates a successful reception of a received CAN Rx L-PDU to the CanIf after passing all filters and validation checks.
<b>Available via</b>	CanIf_Can.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 `CanIfPublicReadRxDpduNotifyStatusApi` and `CanIfRxDpduReadNotifyStatus` 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]** [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.]([SRS\\_BSW\\_00323](#))

**[SWS\_CANIF\_00417]** [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.]([SRS\\_BSW\\_00323](#))

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]** [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.]([SRS\\_BSW\\_00323](#))

**[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).

**[SWS\_CANIF\_00423]** [Configuration of `CanIf_RxIndication()`: Each Rx L-PDU (see `CanIfRxDpduCfg`) has to be configured with a corresponding receive indication service of an upper layer module (see [[SWS\\_CANIF\\_00012](#)]) which is called in `CanIf_RxIndication()`.]()



#### 8.4.4 CanIf\_XLRxIndication

[SWS\_CANIF\_91015] [

<b>Service Name</b>	CanIf_XLRxIndication	
<b>Syntax</b>	<pre>void CanIf_XLRxIndication (     const CanXL_HwType* Mailbox,     const PduInfoType* PduInfoPtr )</pre>	
<b>Service ID [hex]</b>	0x55	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Mailbox	Identifies the HRH and its corresponding CAN Controller
	PduInfoPtr	Pointer to the received L-PDU
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	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.	
<b>Available via</b>	CanIf_Can.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 arbitrary `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).] ([SRS\\_Can\\_02003](#))

[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.] ([SRS\\_Can\\_02003](#))

[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](#)].] ([SRS\\_Can\\_02003](#))

[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`. In case the matching `CanIfRxPduCfg` has configured `CanIfRxPduCfg->CanIfRxPduUserRxIndicationUL` equals CDD, it shall be passed further containing `MetaData` to the configured `<User_RxIndication>()`.] ([SRS\\_Can\\_02003](#))

**[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.] ([SRS\\_Can\\_02003](#))

**[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.] ([SRS\\_Can\\_02003](#))

### 8.4.5 CanIf\_ControllerBusOff

**[SWS\_CANIF\_00218]** [

<b>Service Name</b>	CanIf_ControllerBusOff	
<b>Syntax</b>	void CanIf_ControllerBusOff ( uint8 ControllerId )	
<b>Service ID [hex]</b>	0x16	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ControllerId	Abstract CanIf ControllerId which is assigned to a CAN controller, where a BusOff occurred.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates a Controller BusOff event referring to the corresponding CAN Controller with the abstract CanIf ControllerId.	
<b>Available via</b>	CanIf_Can.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]** [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.] ([SRS\\_BSW\\_00323](#))

**[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.

### 8.4.6 CanIf\_ConfirmPnAvailability

**[SWS\_CANIF\_00815]** [

<b>Service Name</b>	CanIf_ConfirmPnAvailability	
<b>Syntax</b>	void CanIf_ConfirmPnAvailability ( uint8 TransceiverId )	
<b>Service ID [hex]</b>	0x1a	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, which was checked for PN availability.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates that the transceiver is running in PN communication mode referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.	
<b>Available via</b>	CanIf_CanTrcv.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.`.]()

#### 8.4.7 CanIf\_ClearTrcvWufFlagIndication

**[SWS\_CANIF\_00762]** [

<b>Service Name</b>	CanIf_ClearTrcvWufFlagIndication	
<b>Syntax</b>	<pre>void CanIf_ClearTrcvWufFlagIndication (     uint8 TransceiverId )</pre>	
<b>Service ID [hex]</b>	0x20	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, for which this function was called.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates that the transceiver has cleared the WufFlag referring to the corresponding CAN transceiver with the abstract CanIf TransceiverId.	
<b>Available via</b>	CanIf_CanTrcv.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.`.]()

### 8.4.8 CanIf\_CheckTrcvWakeFlagIndication

[SWS\_CANIF\_00763] [

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

### 8.4.9 CanIf\_ControllerModeIndication

[SWS\_CANIF\_00699] [

<b>Service Name</b>	CanIf_ControllerModeIndication	
<b>Syntax</b>	<pre>void CanIf_ControllerModeIndication (     uint8 ControllerId,     Can_ControllerStateType ControllerMode )</pre>	
<b>Service ID [hex]</b>	0x17	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ControllerId	Abstract CanIf ControllerId which is assigned to a CAN controller, which state has been transitioned.
	ControllerMode	Mode to which the CAN controller transitioned
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates a controller state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId.	
<b>Available via</b>	CanIf_Can.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_ControllerModeIndication()` 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).

### 8.4.10 CanIf\_TrvcModeIndication

[SWS\_CANIF\_00764] [

<b>Service Name</b>	CanIf_TrvcModeIndication	
<b>Syntax</b>	<pre>void CanIf_TrvcModeIndication (     uint8 TransceiverId,     CanTrcv_TrvcModeType TransceiverMode )</pre>	





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

]()

Note: The callback service `CanIf_TrvcModeIndication()` 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_TrvcModeIndication()` 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_TrvcModeIndication()` is called.]()

**[SWS\_CANIF\_00708]** [If `CanIf` was not initialized before calling `CanIf_TrvcModeIndication()`, `CanIf` shall not execute state transition notification, when `CanIf_TrvcModeIndication()` is called.]()

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

**[SWS\_CANIF\_00710]** [Configuration of `CanIf_TrvcModeIndication()`: ID of the `CAN Transceiver` is published inside the configuration description of `CanIf` via parameter `CanIfTrcvId`.]()

**[SWS\_CANIF\_00730]** [Configuration of `CanIf_TrvcModeIndication()`: If transceivers are not supported (`CanIfTrcvDrvCfg` is not configured, see `CanIfTrcvDrvCfg`), `CanIf_TrvcModeIndication()` shall not be provided by `CanIf`.]()

### 8.4.11 CanIf\_ControllerErrorStatePassive

[SWS\_CANIF\_91008] [

<b>Service Name</b>	CanIf_ControllerErrorStatePassive	
<b>Syntax</b>	<pre>void CanIf_ControllerErrorStatePassive (     uint8 ControllerId,     uint16 RxErrorCounter,     uint16 TxErrorCounter )</pre>	
<b>Service ID [hex]</b>	0x4f	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	void	–
<b>Description</b>	The function derives the ErrorCounterTreshold from RxErrorCounter/ TxErrorCounter values and reports it to the IdsM as security event CANIF_SEV_ERRORSTATE_PASSIVE to the IdsM. It also prepares the context data for the respective security event.	
<b>Available via</b>	CanIf_Can.h	

](RS\_Ids\_00810)

[SWS\_CANIF\_00919] [If parameter ControllerId of CanIf\_ControllerErrorStatePassive() 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.](RS\_Ids\_00810)

### 8.4.12 CanIf\_ErrorNotification

[SWS\_CANIF\_91009] [

<b>Service Name</b>	CanIf_ErrorNotification	
<b>Syntax</b>	<pre>void CanIf_ErrorNotification (     uint8 ControllerId,     Can_ErrorType Can_ErrorType )</pre>	
<b>Service ID [hex]</b>	0x50	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller.
	Can_ErrorType	Reported CAN error
<b>Parameters (inout)</b>	None	



△

<b>Parameters (out)</b>	None	
<b>Return value</b>	void	-
<b>Description</b>	The function shall derive the bus error source rx or tx from the parameter CanError and report the bus error as security event CANIF_SEV_TX_ERROR_DETECTED or CANIF_SEV_RX_ERROR_DETECTED. It also prepares the context data for the respective security event.	
<b>Available via</b>	CanIf_Can.h	

](RS\_Ids\_00810)

**[SWS\_CANIF\_00920]** [If parameter ControllerId of CanIf\_ErrorNotification() 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.](RS\_Ids\_00810)

**[SWS\_CANIF\_00921]** [If parameter CanError of CanIf\_ErrorNotification() 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.](RS\_Ids\_00810)

## 8.5 Scheduled functions

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

## 8.6 Expected interfaces

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

### 8.6.1 Mandatory interfaces

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



[SWS\_CANIF\_00040] [

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.

]()

## 8.6.2 Optional interfaces

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

[SWS\_CANIF\_00294] [

API Function	Header File	Description
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.
CanNm_RxIndication	CanNm.h	Indication of a received PDU from a lower layer communication interface module.
CanNm_TxConfirmation	CanNm.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.





<b>API Function</b>	<b>Header File</b>	<b>Description</b>
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_ConfirmPnAvailability	CanSM_CanIf.h	This callback function indicates that the transceiver is running in PN communication mode.
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.
CanTp_RxIndication	CanTp.h	Indication of a received PDU from a lower layer communication interface module.
CanTp_TxConfirmation	CanTp.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
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.
CanTSyn_RxIndication	CanTSyn.h	Indication of a received PDU from a lower layer communication interface module.
CanTSyn_TxConfirmation	CanTSyn.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
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.
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_SetSecurityEventWithContext Data	IdsM.h	This API is the application interface to report security events with context data to the IdsM.
J1939Nm_RxIndication	J1939Nm.h	Indication of a received PDU from a lower layer communication interface module.
J1939Nm_TxConfirmation	J1939Nm.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.





API Function	Header File	Description
J1939Tp_RxIndication	J1939Tp.h	Indication of a received PDU from a lower layer communication interface module.
J1939Tp_TxConfirmation	J1939Tp.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.
PduR_CanIfRxIndication	PduR_CanIf.h	Indication of a received PDU from a lower layer communication interface module.
PduR_CanIfTxConfirmation	PduR_CanIf.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
Xcp_CanIfRxIndication	Xcp.h	Indication of a received PDU from a lower layer communication interface module.
Xcp_CanIfTxConfirmation	Xcp.h	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.

⌋()

### 8.6.3 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.

### 8.6.3.1 <User\_TriggerTransmit>

[SWS\_CANIF\_00886] [

<b>Service Name</b>	<User_TriggerTransmit>	
<b>Syntax</b>	Std_ReturnType <User_TriggerTransmit> ( PduIdType TxPduId, PduInfoType* PduInfoPtr )	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant for different Pdulds. Non reentrant for the same Pdul.	
<b>Parameters (in)</b>	TxPdul	ID of the SDU that is requested to be transmitted.
<b>Parameters (inout)</b>	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLenh. On return, the service will indicate the length of the copied SDU data in SduLength.
<b>Parameters (out)</b>	None	
<b>Return value</b>	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.
<b>Description</b>	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.	
<b>Available via</b>	configurable	

]()

Note: This callback service is called by [CanIf](#) and implemented in the corresponding upper layer module. It is called in case of a *Trigger Transmit* request of [CanDrv](#).

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

[SWS\_CANIF\_00888] [Configuration of <User\_TriggerTransmit>(): The upper layer module, which provides the *TriggerTransmit* callback service, has to be configured by [CanIfTxPduUserTxConfirmationUL](#) (see [CanIfTxPduUserTxConfirmationUL](#)). If no upper layer modules are configured, no *TriggerTransmit* callback service is executed and therefore *Trigger Transmit* functionality is not supported for that PDU.]()

[SWS\_CANIF\_00889] [Configuration of <User\_TriggerTransmit>(): The name of the API <User\_TriggerTransmit>() which is called by [CanIf](#) shall be configured for [CanIf](#) by parameter [CanIfTxPduUserTriggerTransmitName](#) (see [CanIfTxPduUserTriggerTransmitName](#)).]()

Note: If [CanIfTxPduTriggerTransmit](#) is not specified or FALSE, no upper layer modules have to be configured for *Trigger Transmit*. Therefore, <User\_TriggerTransmit>() will not be called and [CanIfTxPduUserTxConfirmationUL](#) as well as [CanIfTxPduUserTriggerTransmitName](#) need not to be configured.

**[SWS\_CANIF\_00890]** [Configuration of `<User_TriggerTransmit>()`: If `CanIfTxPduUserTxConfirmationUL` is set to PDUR, `CanIfTxPduUserTriggerTransmitName` must be `PduR_CanIfTriggerTransmit`.]()

**[SWS\_CANIF\_00891]** [Configuration of `<User_TriggerTransmit>()`: If `CanIfTxPduUserTxConfirmationUL` is set to CDD, the name of the API `<User_TriggerTransmit>()` has to be configured via parameter `CanIfTxPduUserTriggerTransmitName`.]()

### 8.6.3.2 <User\_TxConfirmation>

**[SWS\_CANIF\_00011]** [

<b>Service Name</b>	<User_TxConfirmation>	
<b>Syntax</b>	<pre>void &lt;User_TxConfirmation&gt; (     PduIdType TxPduId,     Std_ReturnType result )</pre>	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant for different Pdulds. Non reentrant for the same Pduld.	
<b>Parameters (in)</b>	TxPdulId	ID of the PDU that has been transmitted.
	result	E_OK: The PDU was transmitted. E_NOT_OK: Transmission of the PDU failed.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.	
<b>Available via</b>	configurable	

]()

Note: This callback service is called by `CanIf` and implemented in the corresponding upper layer module. It is called in case of a *transmit confirmation* of `CanDrv`.

Note: This type of confirmation callback service is mainly designed for `PduR`, `CanNm`, and `CanTp`, but not exclusive.

Note: Parameter `TxPduId` is derived from `<User>` configuration.

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

**[SWS\_CANIF\_00438]** [Configuration of `<User_TxConfirmation>()`: The upper layer module, which provides this callback service, has to be configured by `CanIfTxPduUserTxConfirmationUL`. If no upper layer modules are configured for *transmit confirmation* using `<User_TxConfirmation>()`, no *transmit confirmation* is executed.]()

**[SWS\_CANIF\_00542]** [Configuration of `<User_TxConfirmation>()`: The name of the API `<User_TxConfirmation>()` which is called by `CanIf` shall be configured for `CanIf` by parameter `CanIfTxPduUserTxConfirmationName`.]()

Note: If *transmit confirmations* are not necessary or no upper layer modules are configured for *transmit confirmations* and thus `<User_TxConfirmation>()` shall not be called, `CanIfTxPduUserTxConfirmationUL` and `CanIfTxPduUserTxConfirmationName` need not to be configured.

**[SWS\_CANIF\_00439]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `PDUR`, `CanIfTxPduUserTxConfirmationName` must be `PduR_CanIfTxConfirmation`.]()

**[SWS\_CANIF\_00543]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `CAN_NM`, `CanIfTxPduUserTxConfirmationName` must be `CanNm_TxConfirmation`.]()

Hint (Dependency to another module):

If at least one `CanIf Tx L-SDU` is configured with `CanNm_TxConfirmation()`, which means `CanIfTxPduUserTxConfirmationUL` equals `CAN_NM`, the `CanNm` configuration parameter `CANNM_IMMEDIATE_TXCONF_ENABLED` must be set to `FALSE` (for `CanNm` related details see [4, Specification of CAN Network Management], [SWS\_CANNM\_00284]).

**[SWS\_CANIF\_00858]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `J1939NM`, `CanIfTxPduUserTxConfirmationName` must be `J1939Nm_TxConfirmation`.]()

**[SWS\_CANIF\_00544]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `J1939TP`, `CanIfTxPduUserTxConfirmationName` must be `J1939Tp_TxConfirmation`.]()

**[SWS\_CANIF\_00550]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `CAN_TP`, `CanIfTxPduUserTxConfirmationName` must be `CanTp_TxConfirmation`.]()

**[SWS\_CANIF\_00556]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `XCP`, `CanIfTxPduUserTxConfirmationName` must be `Xcp_CanIfTxConfirmation`.]()

**[SWS\_CANIF\_00551]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `CDD`, the name of the API `<User_TxConfirmation>()` has to be configured via parameter `CanIfTxPduUserTxConfirmationName`.]()

**[SWS\_CANIF\_00879]** [Configuration of `<User_TxConfirmation>()`: If `CanIfTxPduUserTxConfirmationUL` is set to `CAN_TSYN`, `CanIfTxPduUserTxConfirmationName` must be `CanTSyn_TxConfirmation`.]()

### 8.6.3.3 <User\_RxIndication>

[SWS\_CANIF\_00012] [

<b>Service Name</b>	<User_RxIndication>	
<b>Syntax</b>	<pre>void &lt;User_RxIndication&gt; (     PduIdType RxPduId,     const PduInfoType* PduInfoPtr )</pre>	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant for different PduIds. Non reentrant for the same PduId.	
<b>Parameters (in)</b>	RxPduId	ID of the received PDU.
	PduInfoPtr	Contains the length (SduLength) of the received PDU, a pointer to a buffer (SduDataPtr) containing the PDU, and the MetaData related to this PDU.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Indication of a received PDU from a lower layer communication interface module.	
<b>Available via</b>	configurable	

] ([SRS\\_Can\\_01003](#))

Note: This service indicates a successful *reception* of an *L-SDU* to the upper layer module after passing all filters and validation checks.

Note: This callback service is called by [CanIf](#) and implemented in the configured upper layer module (e.g. [PduR](#), [CanNm](#), [CanTp](#), etc.) if configured accordingly (see [CanIfRxPduUserRxIndicationUL](#)).

Note: Until [<User\\_RxIndication>\(\)](#) returns, [CanIf](#) will not access [<PduInfoPtr>](#). The [<PduInfoPtr>](#) is only valid and can be used by upper layers, until the indication returns. [CanIf](#) guarantees that the number of configured bytes for this [<PduInfoPtr>](#) is valid.

Note: The call context of [<User\\_RxIndication>\(\)](#) is either on interrupt level (*interrupt mode*) or on task level (*polling mode*).

[SWS\_CANIF\_00441] [Configuration of [<User\\_RxIndication>\(\)](#): The upper layer module, which provides this callback service, has to be configured by [CanIfRxPduUserRxIndicationUL](#).]()

[SWS\_CANIF\_00552] [Configuration of [<User\\_RxIndication>\(\)](#): The name of the API [<User\\_RxIndication>\(\)](#) which will be called by [CanIf](#) shall be configured for [CanIf](#) by parameter [CanIfRxPduUserRxIndicationName](#).]()

Note: If *receive indications* are not necessary or no upper layer modules are configured for *receive indications* and thus [<User\\_RxIndication>\(\)](#) shall not be called, [CanIfRxPduUserRxIndicationUL](#) and [CanIfRxPduUserRxIndicationName](#) need not to be configured.



**[SWS\_CANIF\_00442]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to PDUR, `CanIfRxPduUserRxIndicationName` must be `PduR_CanIfRxIndication.`]()

**[SWS\_CANIF\_00445]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to CAN\_NM, `CanIfRxPduUserRxIndicationName` must be `CanNm_RxIndication.`]()

The value passed to `CanNm` via the API parameter `CanNmRxPduId` refers to the `CanNm` channel handle within the `CanNm` module (for `CanNm` related details see [4, Specification of CAN Network Management]).

**[SWS\_CANIF\_00859]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to J1939NM, `CanIfRxPduUserRxIndicationName` must be `J1939Nm_RxIndication.`]()

**[SWS\_CANIF\_00448]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to CAN\_TP, `CanIfRxPduUserRxIndicationName` must be `CanTp_RxIndication.`]()

**[SWS\_CANIF\_00554]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to J1939TP, `CanIfRxPduUserRxIndicationName` must be `J1939Tp_RxIndication.`]()

**[SWS\_CANIF\_00555]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to XCP, `CanIfRxPduUserRxIndicationName` must be `Xcp_CanIfRxIndication.`]()

**[SWS\_CANIF\_00557]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to CDD the name of the API has to be configured via parameter `CanIfRxPduUserRxIndicationName.`]()

**[SWS\_CANIF\_00880]** [Configuration of `<User_RxIndication>()`: If `CanIfRxPduUserRxIndicationUL` is set to CAN\_TSYN, `CanIfRxPduUserRxIndicationName` must be `CanTSyn_RxIndication.`]()

### 8.6.3.4 `<User_ValidateWakeupEvent>`

**[SWS\_CANIF\_00532]** [

<b>Service Name</b>	<code>&lt;User_ValidateWakeupEvent&gt;</code>	
<b>Syntax</b>	<pre>void &lt;User_ValidateWakeupEvent&gt; (     EcuM_WakeupSourceType sources )</pre>	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant (defined within providing upper layer module)	
<b>Parameters (in)</b>	<code>sources</code>	Validated CAN wakeup events. Every CAN controller or CAN transceiver can be a separate wakeup source.







<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	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.
<b>Available via</b>	configurable

]()

Note: This callback service is mainly implemented in and used by the *ECU State Manager* module (see [13, 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 re-entrant 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`.]()

### 8.6.3.5 <User\_ControllerBusOff>

[SWS\_CANIF\_00014] [

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

]([SRS\\_Can\\_01029](#))

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`.]

### 8.6.3.6 `<User_ConfirmPnAvailability>`

**[SWS\_CANIF\_00821]** [

<b>Service Name</b>	<code>&lt;User_ConfirmPnAvailability&gt;</code>	
<b>Syntax</b>	<pre>void &lt;User_ConfirmPnAvailability&gt; (     uint8 TransceiverId )</pre>	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant (defined within providing upper layer module)	
<b>Parameters (in)</b>	TransceiverId	Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, which was checked for PN availability.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates that the CAN transceiver is running in PN communication mode.	
<b>Available via</b>	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_ConfirmPnAvailability>()` is either on interrupt level (interrupt mode) or on task level (polling mode).

Note: The callback service `<User_ConfirmPnAvailability>()` is in general re-entrant 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`.]

**[SWS\_CANIF\_00824]** [Configuration of `<User_ConfirmPnAvailability>()`: The name of `<User_ConfirmPnAvailability>()` shall be configurable by `CanIfDispatchUserConfirmPnAvailabilityName` if `CanIfPublicPnSupport` equals `True`.]

**[SWS\_CANIF\_00825]** [Configuration of `<User_ConfirmPnAvailability>()`: It shall be configurable by `CanIfPublicPnSupport`, 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`.]()

### 8.6.3.7 `<User_ClearTrcvWufFlagIndication>`

**[SWS\_CANIF\_00788]** [

<b>Service Name</b>	<code>&lt;User_ClearTrcvWufFlagIndication&gt;</code>	
<b>Syntax</b>	<pre>void &lt;User_ClearTrcvWufFlagIndication&gt; (     uint8 TransceiverId )</pre>	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	TransceiverId	Abstracted CanIf TransceiverId, for which this function was called.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates that the CAN transceiver has cleared the WufFlag. This function is called in <code>CanIf_ClearTrcvWufFlagIndication</code> .	
<b>Available via</b>	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_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`.]()

**[SWS\_CANIF\_00795]** [Configuration of <User\_ClearTrcvWufFlagIndication>(): The name of <User\_ClearTrcvWufFlagIndication>() shall be configurable by [CanIfDispatchUserClearTrcvWufFlagIndicationName](#) if [CanIfPublicPnSupport](#) equals True.]()

**[SWS\_CANIF\_00796]** [Configuration of <User\_ClearTrcvWufFlagIndication>(): It shall be configurable by [CanIfPublicPnSupport](#), 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](#).]()

### 8.6.3.8 <User\_CheckTrcvWakeFlagIndication>

**[SWS\_CANIF\_00814]** [

<b>Service Name</b>	<User_CheckTrcvWakeFlagIndication>	
<b>Syntax</b>	void <User_CheckTrcvWakeFlagIndication> ( uint8 TransceiverId )	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	TransceiverId	Abstracted CanIf TransceiverId, for which this function was called.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates that the wake up flag in the CAN transceiver is set. This function is called in CanIf_CheckTrcvWakeFlagIndication.	
<b>Available via</b>	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\_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`.]()

**[SWS\_CANIF\_00801]** [Configuration of `<User_CheckTrcvWakeFlagIndication>()`: The name of `<User_CheckTrcvWakeFlagIndication>()` shall be configurable by `CanIfDispatchUserCheckTrcvWakeFlagIndicationName` if `CanIfPublicPnSupport` equals `True`.]()

**[SWS\_CANIF\_00802]** [Configuration of `<User_CheckTrcvWakeFlagIndication>()`: It shall be configurable by `CanIfPublicPnSupport`, 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`.]()

### 8.6.3.9 `<User_ControllerModeIndication>`

**[SWS\_CANIF\_00687]** [

<b>Service Name</b>	<code>&lt;User_ControllerModeIndication&gt;</code>	
<b>Syntax</b>	<pre>void &lt;User_ControllerModeIndication&gt; (     uint8 ControllerId,     Can_ControllerStateType ControllerMode )</pre>	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, at which a controller state transition occurred.
	ControllerMode	Notified CAN controller mode
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates a CAN controller state transition to the corresponding upper layer module (mainly the CAN State Manager module).	
<b>Available via</b>	configurable	

]()

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\_00692]** [Configuration of `<User_ControllerModeIndication>()`]: If `CanIfDispatchUserCtrlModeIndicationUL` is set to `CDD` the name of the function has to be configured via parameter `CanIfDispatchUserCtrlModeIndicationName.()`

### 8.6.3.10 <User\_TrcvModeIndication>

**[SWS\_CANIF\_00693]** [

<b>Service Name</b>	<code>&lt;User_TrcvModeIndication&gt;</code>
<b>Syntax</b>	<pre>void &lt;User_TrcvModeIndication&gt; (     uint8 TransceiverId,     CanTrcv_TrcvModeType TransceiverMode )</pre>
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant







<b>Parameters (in)</b>	TransceiverId	Abstracted CanIf TransceiverId which is assigned to a CAN transceiver, at which a transceiver state transition occurred.
	TransceiverMode	Notified CAN transceiver mode
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This service indicates a CAN transceiver state transition to the corresponding upper layer module (mainly the CAN State Manager module).	
<b>Available via</b>	configurable	

]()

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\\_TrcevModeIndication\(\)](#) of the [CanTrcv](#). The delivered parameter `Transceiver` of the service [CanIf\\_TrcevModeIndication\(\)](#) 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\\_TrcevModeIndication\(\)](#) is mapped to the appropriate parameter `TransceiverMode` of `<User_TrcevModeIndication>()`.

Note: For different upper layer users different service names shall be used.

**[SWS\_CANIF\_00694]** [Caveats of `<User_TrcevModeIndication>()`]:

- 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_TrcevModeIndication>()`]: The upper layer module which provides this callback service has to be configured by [CanIfDispatchUserTrcevModeIndicationUL](#), but:

- If no upper layer modules are configured for *transceiver mode indications* using `<User_TrcevModeIndication>()`, no *transceiver mode indication* needs to be configured. [CanIfDispatchUserTrcevModeIndicationUL](#) needs not to be configured.
- If transceivers are not supported ([CanIfTrcvDrvCfg](#) is not configured, [CanIfDispatchUserTrcevModeIndicationUL](#) is not configurable).

]()



If no upper layer modules are configured for *state transition notifications* using `<User_TrvcModeIndication>()`, no *state transition notification* needs to be configured.

**[SWS\_CANIF\_00696]** [Configuration of `<User_TrvcModeIndication>()`: The name of `<User_TrvcModeIndication>()` which will be called by `CanIf` shall be configured for `CanIf` by parameter `CanIfDispatchUserTrvcModeIndicationName`. This is only necessary if *state transition notifications* are configured via `CanIfDispatchUserTrvcModeIndicationUL`.]()

**[SWS\_CANIF\_00697]** [Configuration of `<User_TrvcModeIndication>()`: If `CanIfDispatchUserTrvcModeIndicationUL` is set to `CAN_SM`, `CanIfDispatchUserTrvcModeIndicationName` must be `CanSM_TransceiverModeIndication`.]()

**[SWS\_CANIF\_00698]** [Configuration of `<User_TrvcModeIndication>()`: If `CanIfDispatchUserTrvcModeIndicationUL` is set to `CDD` the name of the API has to be configured via parameter `CanIfDispatchUserTrvcModeIndicationName`.]()

## 9 Sequence diagrams

The following sequence diagrams show the interactions between `CanIf` and `CanDrv`.

### 9.1 Transmit request (single CAN Driver)

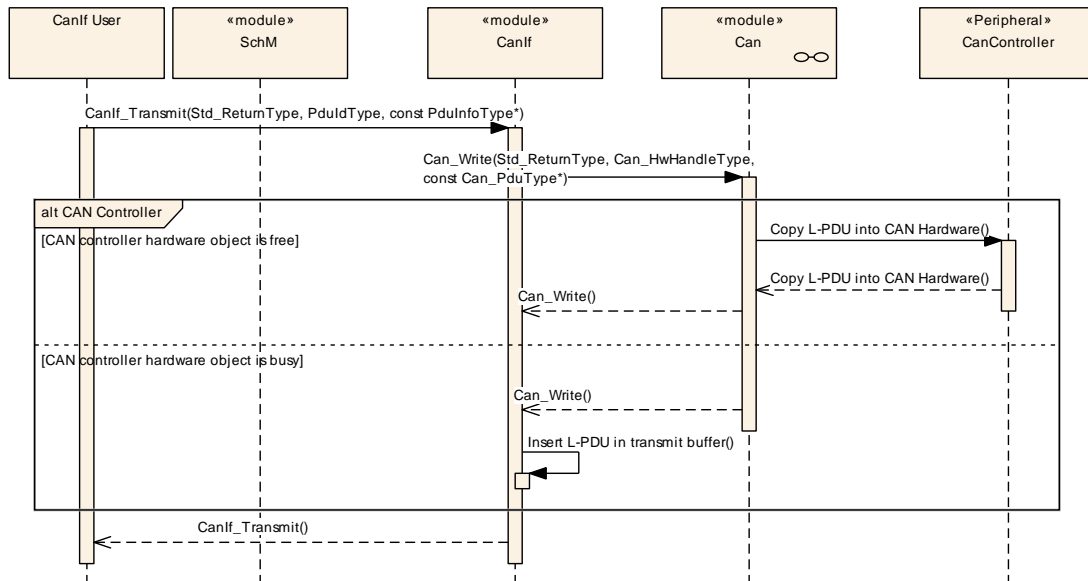


Figure 9.1: Transmission request with a single CAN Driver

Activity	Description
<b>Transmission request</b>	The upper layer initiates a transmit request via the service <code>CanIf_Transmit()</code> . The parameter <code>CanTxPduId</code> identifies the requested L-SDU. The service performs following steps: <ul style="list-style-type: none"> <li>validation of the input parameter</li> <li>definition of the <code>CAN Controller</code> to be used</li> </ul> The second parameter <code>*PduInfoPtr</code> is a pointer on the structure with transmit L-SDU related data such as <code>SduLength</code> and <code>*SduDataPtr</code> .
<b>Start transmission</b>	<code>CanIf_Transmit()</code> requests a transmission and calls the <code>CanDrv</code> service <code>Can_Write()</code> with corresponding processing of the <code>HTH</code> .
<b>Hardware request</b>	<code>Can_Write()</code> writes all L-PDU data in the <code>CAN Hardware</code> (if it is free) and sets the hardware request for transmission.
<b>E_OK from Can_Write service</b>	<code>Can_Write()</code> returns <code>E_OK</code> to <code>CanIf_Transmit()</code> .
<b>CAN_BUSY from Can_Write service</b>	If <code>CanDrv</code> detects, there are no free hardware objects available, it returns <code>CAN_BUSY</code> to <code>CanIf</code> .
<b>Copying into the buffer</b>	The L-PDU of the rejected transmit request will be inserted in the transmit buffer of <code>CanIf</code> until the next transmit confirmation.
<b>E_OK from CanIf</b>	<code>CanIf_Transmit()</code> returns <code>E_OK</code> to the upper layer.

## 9.2 Transmit request (multiple CAN Drivers)

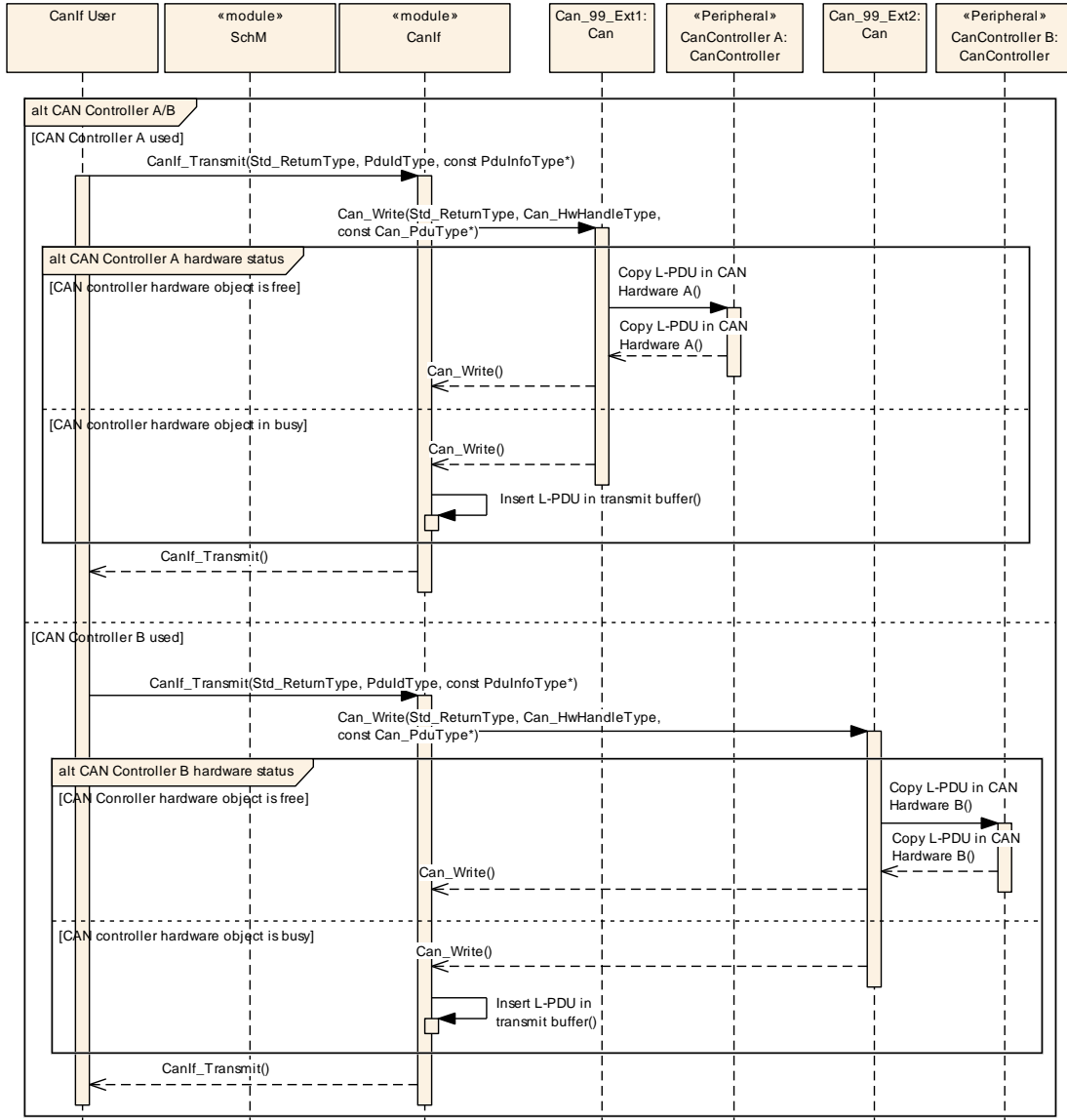


Figure 9.2: Transmission request with multiple CAN Drivers

First transmit request:

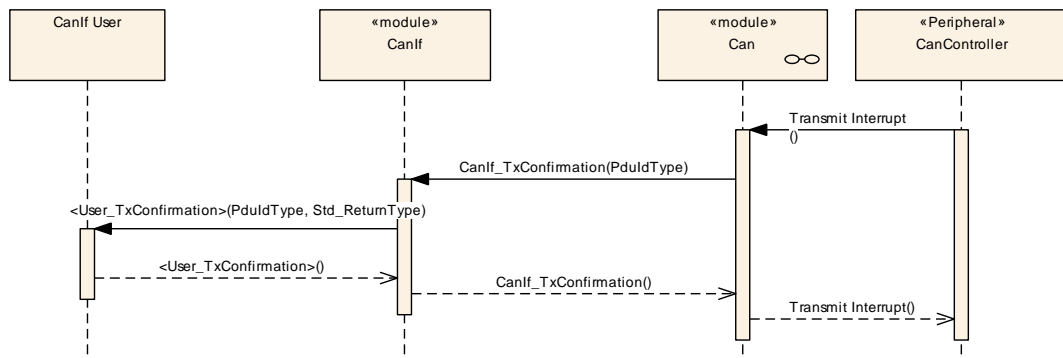
Activity	Description
Transmission request A	<p>The upper layer initiates a transmit request via the service <code>CanIf_Transmit()</code>. The parameter <code>CanTxPduId</code> identifies the requested L-SDU. The service performs following steps:</p> <ul style="list-style-type: none"> <li>validation of the input parameter</li> <li>definition of the CAN Controller to be used (here: <code>Can_99_Ext1</code>)</li> </ul> <p>The second parameter <code>*PduInfoPtr</code> is a pointer on the structure with transmit L-SDU related data such as <code>SduLength</code> and <code>*SduDataPtr</code>.</p>

<b>Start transmission</b>	<code>CanIf_Transmit()</code> requests a transmission and calls the <code>CanDrv Can_99_Ext1</code> service <code>Can_Write_99_Ext1()</code> with corresponding processing of the <code>HTH</code> .
<b>Hardware request</b>	<code>Can_Write_99_Ext1()</code> writes all <code>L-PDU</code> data in the <code>CAN Hardware</code> of Controller A (if it is free) and sets the hardware request for transmission.
<b>E_OK from Can_Write service</b>	<code>Can_Write_99_Ext1()</code> returns <code>E_OK</code> to <code>CanIf_Transmit()</code> .
<b>CAN_BUSY from Can_Write service</b>	If <code>CanDrv Can_99_Ext1</code> detects, there are no free hardware objects available, it returns <code>CAN_BUSY</code> to <code>CanIf</code> .
<b>Copying into the buffer</b>	The <code>L-PDU</code> of the rejected transmit request will be inserted in the transmit buffers of <code>CanIf</code> until the next transmit confirmation.
<b>E_OK from CanIf</b>	<code>CanIf_Transmit()</code> returns <code>E_OK</code> to the upper layer.

Second transmit request:

Activity	Description
<b>Transmission request B</b>	<p>The upper layer initiates a transmit request via the service <code>CanIf_Transmit()</code>. The parameter <code>CanTxPduId</code> identifies the requested <code>L-SDU</code>. The service performs following steps:</p> <ul style="list-style-type: none"> <li>validation of the input parameter</li> <li>definition of the <code>CAN Controller</code> to be used (here: <code>Can_99_Ext2</code>)</li> </ul> <p>The second parameter <code>*PduInfoPtr</code> is a pointer on the structure with transmit <code>L-SDU</code> related data such as <code>SduLength</code> and <code>*SduDataPtr</code>.</p>
<b>Start transmission</b>	<code>CanIf_Transmit()</code> starts a transmission and calls the <code>CanDrv Can_99_Ext2</code> service <code>Can_Write_99_Ext2()</code> with corresponding processing of the <code>HTH</code> .
<b>Hardware request</b>	<code>Can_Write_99_Ext2()</code> writes all <code>L-PDU</code> data in the <code>CAN Hardware</code> of Controller B (if it is free) and sets the hardware request for transmission.
<b>E_OK from Can_Write service</b>	<code>Can_Write_99_Ext2()</code> returns <code>E_OK</code> to <code>CanIf_Transmit()</code> .
<b>CAN_BUSY from Can_Write service</b>	If <code>CanDrv Can_99_Ext2</code> detects, there are no free hardware objects available, it returns <code>CAN_BUSY</code> to <code>CanIf</code> .
<b>Copying into the buffer</b>	The <code>L-PDU</code> of the rejected transmit request will be inserted in the transmit buffers of <code>CanIf</code> until the next transmit confirmation.
<b>E_OK from CanIf</b>	<code>CanIf_Transmit()</code> returns <code>E_OK</code> to the upper layer.

### 9.3 Transmit confirmation (interrupt mode)



**Figure 9.3: Transmit confirmation interrupt driven**

Activity	Description
<b>Transmit interrupt</b>	The acknowledged CAN frame signals a successful transmission to the receiving <b>CAN Controller</b> and triggers the transmit interrupt.
<b>Confirmation to CanIf</b>	<b>CanDrv</b> calls the service <b>CanIf_TxConfirmation()</b> . The parameter <b>CanTxPduId</b> specifies the <b>L-PDU</b> previously sent by <b>Can_Write()</b> . <b>CanDrv</b> must store the all in <b>HTHs</b> pending <b>L-PDU</b> Ids in an array organized per <b>HTH</b> to avoid new search of the <b>L-PDU</b> ID for call of <b>CanIf_TxConfirmation()</b> .
<b>Confirmation to upper layer</b>	Calling of the corresponding upper layer confirmation service <b>&lt;User_TxConfirmation&gt;(id, E_OK)</b> . It signals a successful <b>L-SDU</b> transmission to the upper layer.

## 9.4 Transmit confirmation (polling mode)

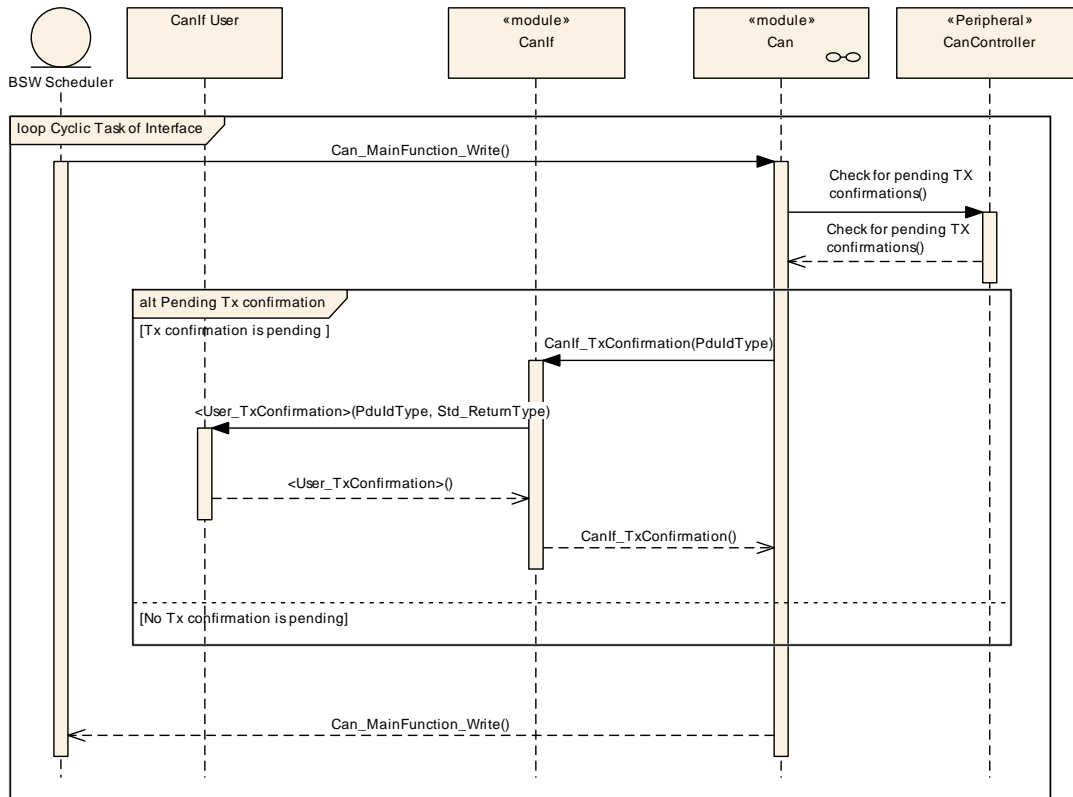
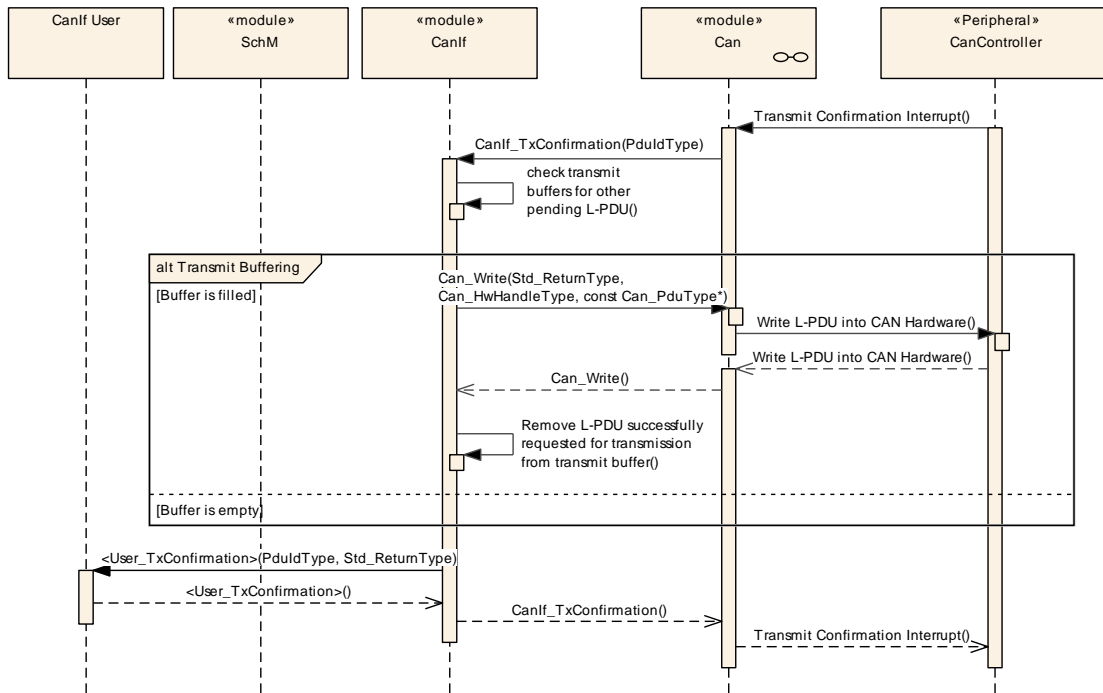


Figure 9.4: Transmit confirmation polling driven

Activity	Description
<b>Cyclic Task <code>CanDrv</code></b>	The service <code>Can_MainFunction_Write()</code> is called by the BSW Scheduler.
<b>Check for pending transmit confirmations</b>	<code>Can_MainFunction_Write()</code> checks the underlying <b>CAN Controller</b> (s) about pending transmit confirmations of previously succeeded transmit events.
<b>Transmit Confirmation</b>	The acknowledged CAN frame signals a successful transmission to the sending <b>CAN Controller</b> .
<b>Confirmation to <code>CanIf</code></b>	<code>CanDrv</code> calls the service <code>CanIf_TxConfirmation()</code> . The parameter <code>CanTxPduId</code> specifies the <b>L-PDU</b> previously sent by <code>Can_Write()</code> . <code>CanDrv</code> must store the all in <b>HTHs</b> pending <b>L-PDU</b> Ids in an array organized per <b>HTH</b> to avoid new search of the <b>L-PDU</b> ID for call of <code>CanIf_TxConfirmation()</code> .
<b>Confirmation to upper layer</b>	Calling of the corresponding upper layer confirmation service <code>&lt;User_TxConfirmation&gt;(id, E_OK)</code> . It signals a successful <b>L-SDU</b> transmission to the upper layer.

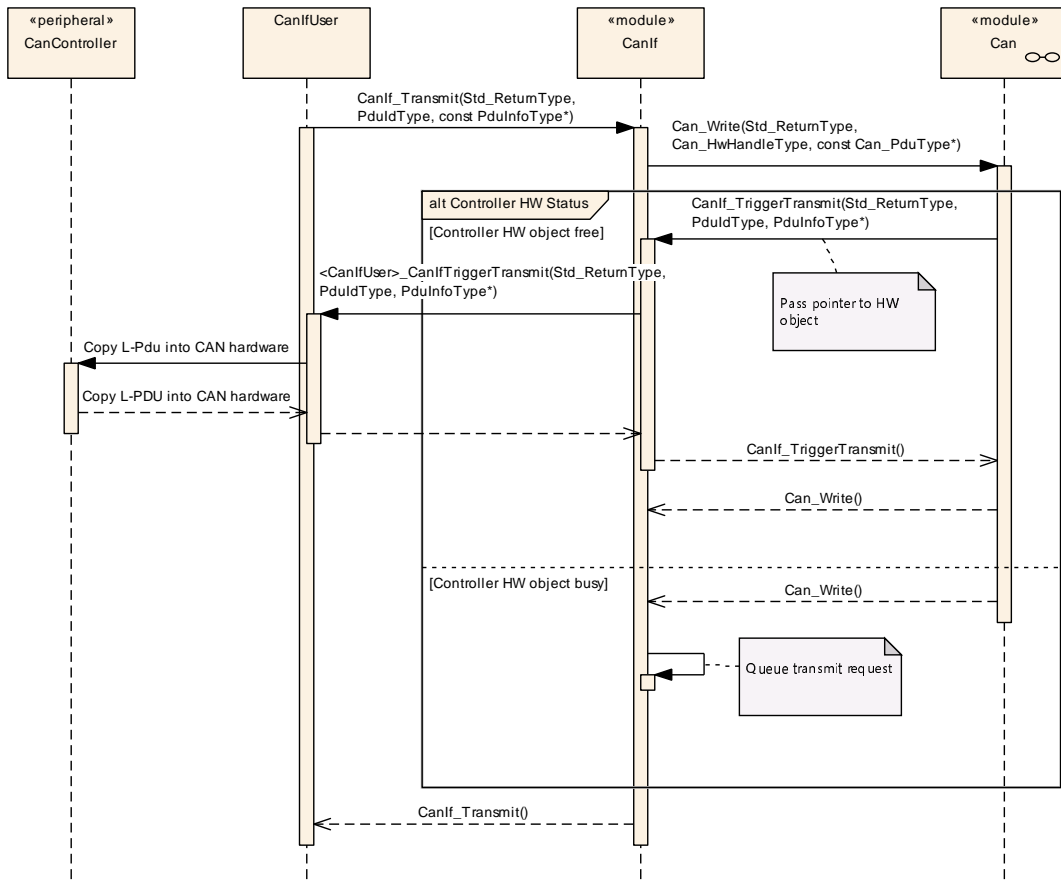
### 9.5 Transmit confirmation (with buffering)



**Figure 9.5: Transmit confirmation with buffering**

Activity	Description
<b>Transmit interrupt</b>	Acknowledged CAN frame signals successful transmission to receiving <b>CAN Controller</b> and triggers transmit interrupt.
<b>Confirmation to CanIf</b>	<b>CanDrv</b> calls service <b>CanIf_TxConfirmation()</b> . Parameter <b>CanTxPduId</b> specifies the <b>L-PDU</b> previously transmitted by <b>Can_Write()</b> . <b>CanDrv</b> must store the all in <b>HTHs</b> pending <b>L-PDU</b> Ids in an array organized per <b>HTH</b> to avoid new search of the <b>L-PDU</b> ID for call of <b>CanIf_TxConfirmation()</b> .
<b>Check of transmit buffers</b>	The transmit buffers of <b>CanIf</b> checked, whether a pending <b>L-PDU</b> is stored or not.
<b>Transmit request passed to CanDrv</b>	In case of pending <b>L-PDU</b> s in the transmit buffers the highest priority order the latest <b>L-PDU</b> is requested for transmission by <b>Can_Write()</b> . It signals a successful <b>L-PDU</b> transmission to the upper layer. Thus <b>Can_Write()</b> can be called re-entrant.
<b>Remove transmitted L-PDU from transmit buffers</b>	The <b>L-PDU</b> pending for transmission is removed from the transmission buffers by <b>CanIf</b> .
<b>Confirmation to the upper layer</b>	Calling of the corresponding upper layer confirmation service <b>&lt;User_TxConfirmation&gt;(id, E_OK)</b> . It signals a successful <b>L-SDU</b> transmission to the upper layer.

## 9.6 Trigger Transmit Request

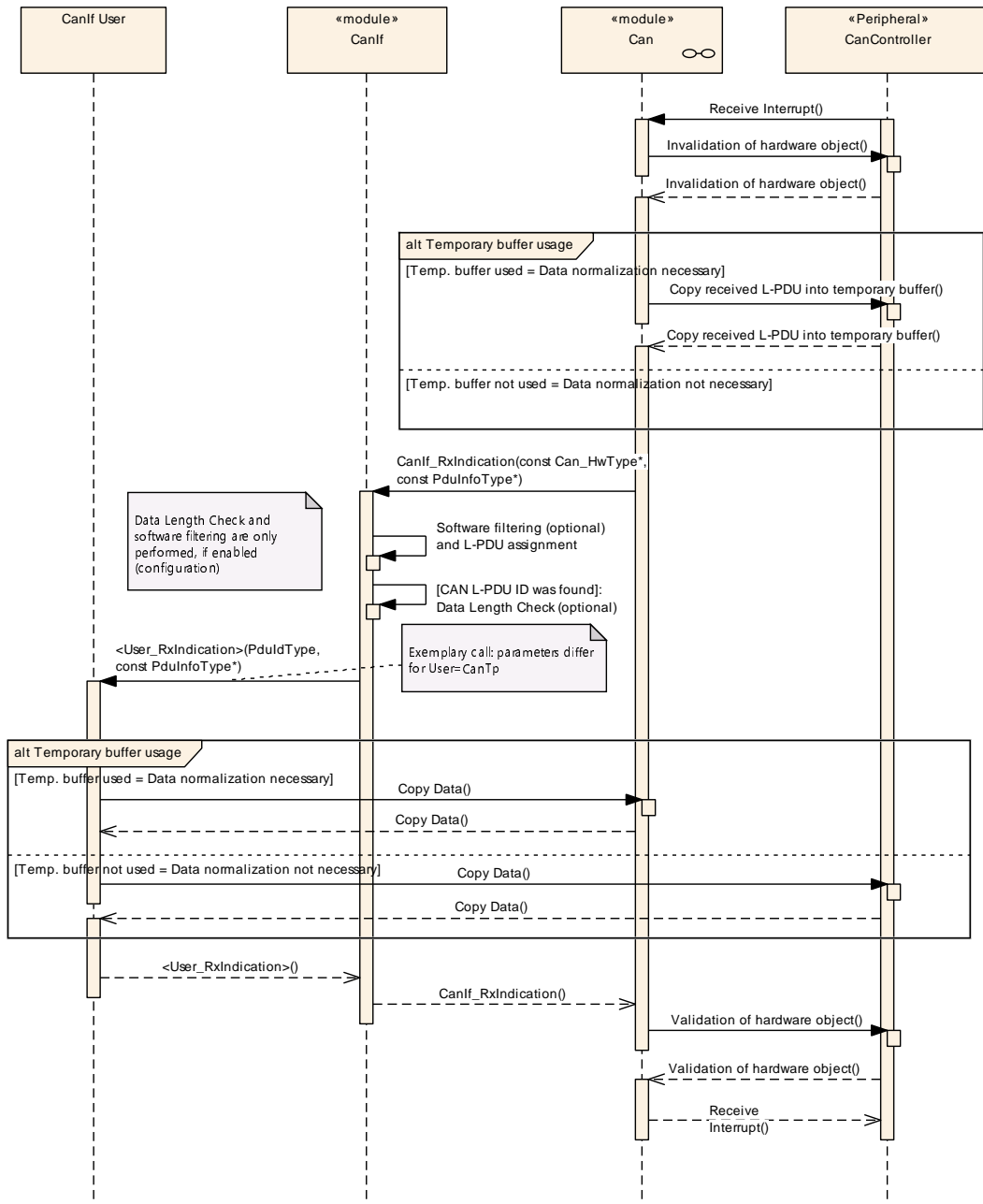


**Figure 9.6: Trigger Transmit Request**



Activity	Description
<b>Transmission request</b>	The upper layer initiates a transmit request via the service <code>CanIf_Transmit()</code> . The parameter <code>CanTxPduId</code> identifies the requested L-SDU. The service performs following steps: <ul style="list-style-type: none"> <li>• validation of the input parameter</li> <li>• definition of the CAN Controller to be used</li> </ul> The second parameter <code>*PduInfoPtr</code> is a pointer to the structure with the size ( <code>SduLength</code> ) of the L-SDU to be transmitted. The actual SDU data has not been passed by the upper layer. Hence, the pointer <code>*SduDataPtr</code> points to NULL.
<b>Start transmission</b>	<code>CanIf_Transmit()</code> requests a transmission and calls the <code>CanDrv</code> service <code>Can_Write()</code> with corresponding processing of the HTH.
<b>Trigger transmission</b>	If the CAN hardware is free <code>Can_Write()</code> requests the SDU data from <code>CanIf</code> by its service <code>CanIf_TriggerTransmit()</code> passing the L-SDUs corresponding ID and a pointer to the CAN hardware's buffer. <code>CanIf</code> forwards the trigger transmit request to the corresponding upper layer ( <code>CanIfUser</code> ). <code>CanIf</code> passes the buffer pointer received by <code>CanDrv</code> . The <code>CanIfUser</code> finally copies the SDU data to the buffer provided by <code>CanIf</code> (the CAN hardware buffer) and returns status and number of bytes effectively written.
<b>E_OK from <code>Can_Write()</code> service</b>	<code>Can_Write()</code> returns E_OK to <code>CanIf_Transmit()</code> .
<b>CAN_BUSY from <code>Can_Write()</code> service</b>	If <code>CanDrv</code> detects, there are no free hardware objects available, it returns CAN_BUSY to <code>CanIf</code> .
<b>Queuing of transmission request</b>	The <code>Transmit Request</code> for the L-PDU, which has been rejected by <code>CanDrv</code> , is queued by <code>CanIf</code> until the next transmit confirmation.
<b>E_OK from <code>CanIf</code></b>	<code>CanIf_Transmit()</code> returns E_OK to the upper layer.

### 9.7 Receive indication (interrupt mode)

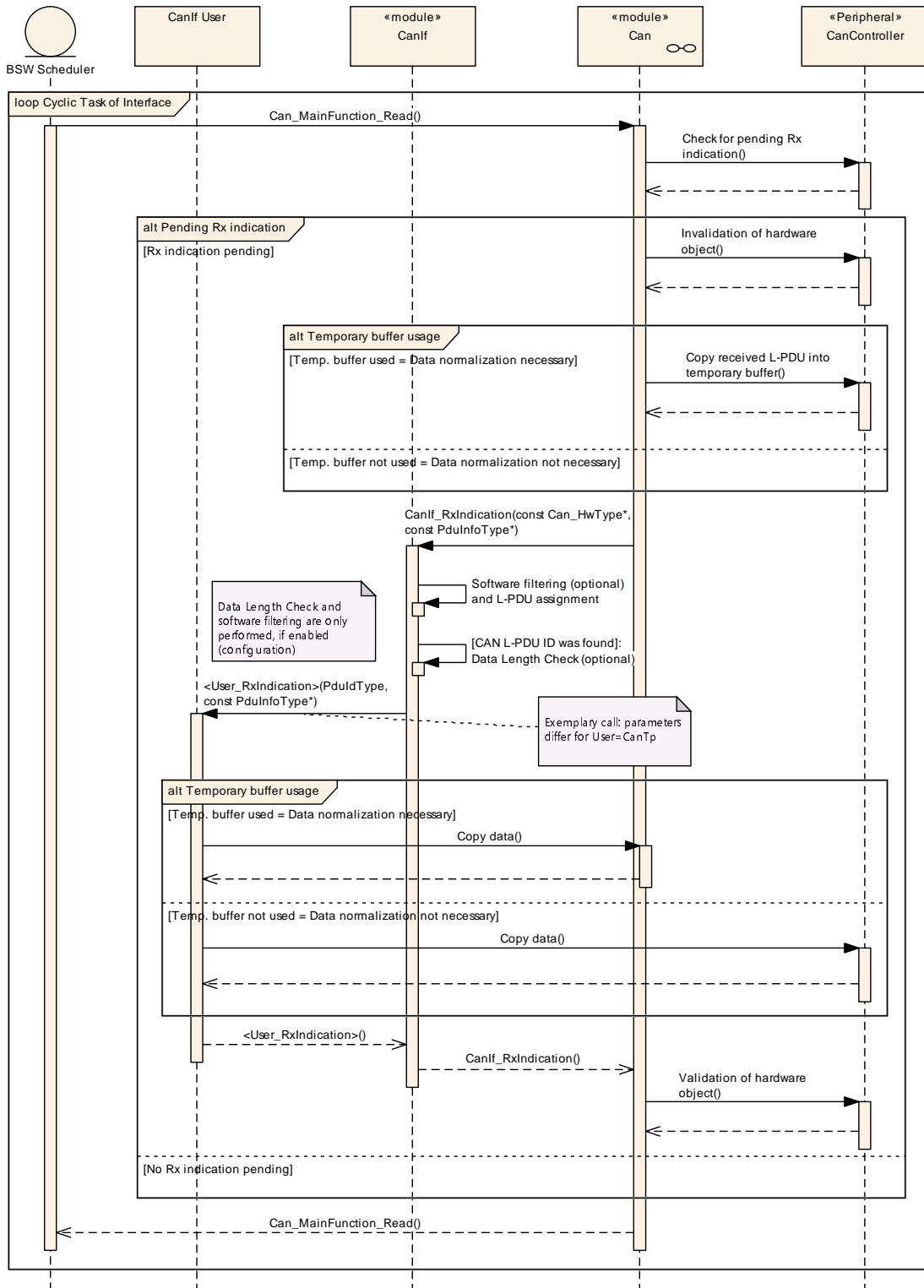


**Figure 9.7: Receive indication interrupt driven**

Activity	Description
<b>Receive Interrupt</b>	The <b>CAN Controller</b> indicates a successful reception and triggers a receive interrupt.
<b>Invalidation of CAN hardware object, provide CPU access to CAN mailbox</b>	The CPU ( <b>CanDrv</b> ) get exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received.

<b>Buffering, normalizing</b>	<p>The L-PDU is normalized and is buffered in the temporary buffer located in <code>CanDrv</code>. Each <code>CanDrv</code> owns such a temporary buffer for every <code>Physical Channel</code> only if normalizing of the data is necessary.</p>
<b>Indication to <code>CanIf</code></b>	<p>The reception is indicated to <code>CanIf</code> by calling of <code>CanIf_RxIndication()</code>. The <code>HRH</code> specifies the <code>CAN RAM Hardware Object</code> and the corresponding <code>CAN Controller</code>, which contains the received L-PDU. The temporary buffer is referenced to <code>CanIf</code> by <code>PduInfoPtr-&gt;SduDataPtr</code>.</p>
<b>Software Filtering</b>	<p>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.</p>
<b>Data Length Check</b>	<p>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.</p>
<b>Receive Indication to the upper layer</b>	<p>The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter <code>RxPduId</code> 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 <code>CAN Controller</code> access.</p>
<b>Validation of CAN hardware object, allow access of <code>CAN Controller</code> to CAN mailbox</b>	<p>The <code>CAN Controller</code> 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.</p>

### 9.8 Receive indication (polling mode)



**Figure 9.8: Receive indication polling driven**

Activity	Description
<b>Cyclic Task <code>CanDrv</code></b>	The service <code>Can_MainFunction_Read()</code> is called by the BSW Scheduler.
<b>Check for new received L-PDU</b>	<code>Can_MainFunction_Read()</code> checks the underlying <code>CAN Controller(s)</code> about new received L-PDUs.
<b>Invalidation of CAN hardware object, provide CPU access to CAN mailbox</b>	In case of a new receive event the CPU ( <code>CanDrv</code> ) get exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received.
<b>Buffering, normalizing</b>	In case of a new receive event the L-PDU is normalized and is buffered in the temporary buffer located in <code>CanDrv</code> . Each <code>CanDrv</code> owns such a temporary buffer for every <code>Physical Channel</code> only if normalizing of the data is necessary.
<b>Indication to <code>CanIf</code></b>	The reception is indicated to <code>CanIf</code> by calling of <code>CanIf_RxIndication()</code> . The HRH specifies the CAN RAM Hardware Object and the corresponding <code>CAN Controller</code> , which contains the received L-PDU. The temporary buffer is referenced to <code>CanIf</code> by <code>PduInfoPtr-&gt;SduDataPtr</code> .
<b>Software Filtering</b>	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.
<b>Data Length Check</b>	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.
<b>Receive Indication to the upper layer</b>	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 <code>RxPduId</code> 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 <code>CAN Controller</code> access.
<b>Validation of CAN hardware object, allow access of <code>CAN Controller</code> to CAN mailbox</b>	The <code>CAN Controller</code> 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

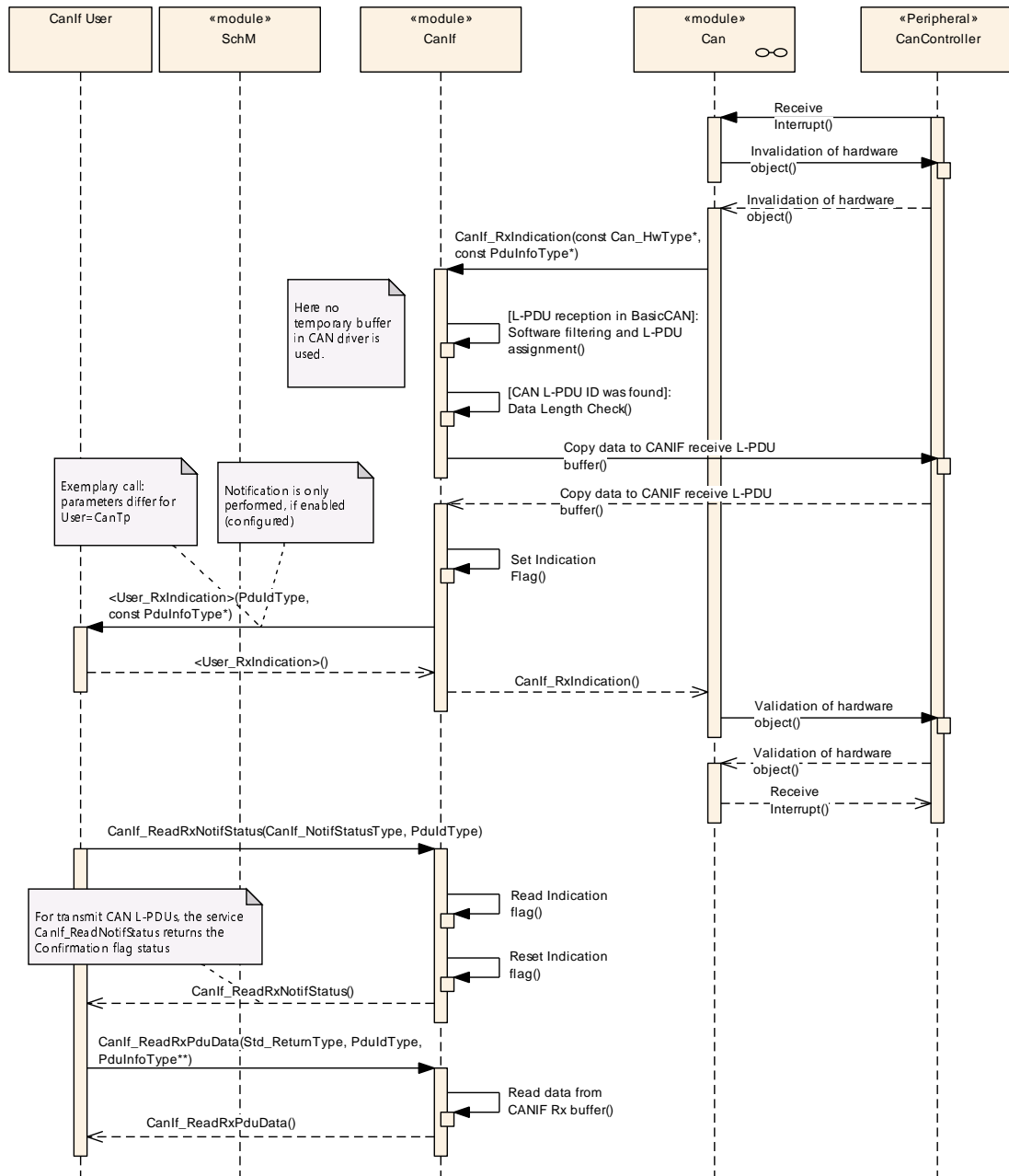
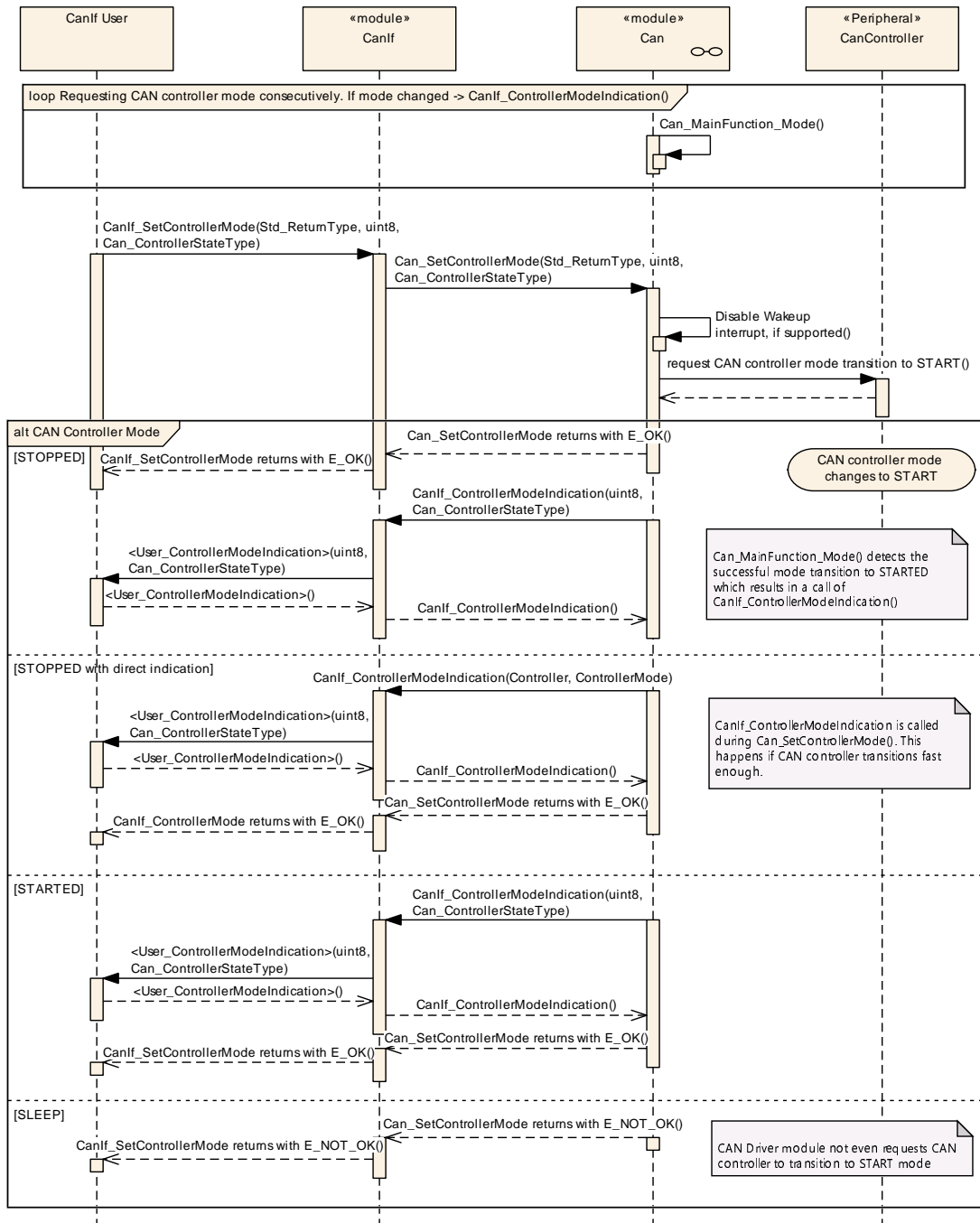


Figure 9.9: Read received data

Activity	Description
<b>Receive Interrupt</b>	The <b>CAN Controller</b> indicates a successful reception and triggers a receive interrupt.
<b>Invalidation of CAN hardware object, provide CPU access to CAN mailbox</b>	The CPU ( <b>CanDrv</b> ) get exclusive access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received.

<b>Buffering, normalizing</b>	The L-PDU is normalized and is buffered in the temporary buffer located in <code>CanDrv</code> . Each <code>CanDrv</code> owns such a temporary buffer for every <code>Physical Channel</code> only if normalizing of the data is necessary.
<b>Indication to <code>CanIf</code></b>	The reception is indicated to <code>CanIf</code> by calling of <code>CanIf_RxIndication()</code> . The <code>HRH</code> specifies the <code>CAN RAM Hardware Object</code> and the corresponding <code>CAN Controller</code> , which contains the received L-PDU. The temporary buffer is referenced to <code>CanIf</code> by <code>PduInfoPtr-&gt;SduDataPtr</code> .
<b>Software Filtering</b>	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.
<b>Data Length Check</b>	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.
<b>Copy data</b>	The data is copied out of the CAN hardware into the receive <code>CAN L-PDU</code> buffers in <code>CanIf</code> . During access the CAN hardware buffers must be unlocked for CPU access/locked for <code>CAN Controller</code> access.
<b>Indication Flag</b>	Set indication status flag for the received L-PDU in <code>CanIf</code> .
<b>Receive Indication to the upper layer</b>	The corresponding receive indication service of the upper layer is called. This signals a successful reception to the target upper layer. The parameter <code>RxPduId</code> specifies the L-SDU, the second parameter is the reference on the temporary buffer within the L-SDU.
<b>Validation of CAN hardware object, allow access of <code>CAN Controller</code> to CAN mailbox</b>	The <code>CAN Controller</code> 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.
<b>Read indication status</b>	Times later the upper layer can read the indication status by call of <code>CanIf_ReadRxNotifStatus()</code> . This service can also be used for transmit L-PDUs. Then it return the confirmation status.
<b>Reset indication status</b>	Before <code>CanIf_ReadRxNotifStatus()</code> returns, the indication status is reset.
<b>Read received data</b>	Times later the upper layer can read the received data by call of <code>CanIf_ReadRxPduData()</code> .
<b>Read <code>CanIf</code> Rx buffer</b>	<code>CanIf_ReadRxPduData()</code> reads the data from <code>CanIf</code> Rx buffer.
<b>E_OK from <code>CanIf</code></b>	If <code>CanIf_ReadRxPduData()</code> was successful, the request returns E_OK with valid <code>PduInfoPtr</code> .

### 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).



<b>The upper layer requests "STARTED" mode of the desired CAN controller</b>	The upper layer calls <code>CanIf_SetControllerMode (ControllerId, CAN_CS_STARTED)</code> to request STARTED mode for the requested CAN controller.
<b>CanDrv disables wake up interrupts, if supported</b>	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.
<b>CanDrv requests the CAN controller to transition into the requested mode ( CAN_CS_STARTED).</b>	During function call <code>Can_SetControllerMode (Controller, Can_ControllerStateType)</code> , 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	
<b>CAN controller was in STOPPED mode</b>	The former request <code>Can_SetControllerMode ()</code> returns and informs CanIf about a successful request which in turn returns the upper layer request <code>CanIf_SetControllerMode ()</code> . The <code>Can_MainFunction_Mode ()</code> detects the successful mode transition of the CAN controller and inform the CanIf asynchronously via <code>CanIf_ControllerModeIndication (Controller, CAN_CS_STARTED)</code> .
<b>CAN controller was in STOPPED mode and the CAN controller transitions very fast so that mode indication is called during transition request</b>	During the former request <code>Can_SetControllerMode ()</code> the function <code>CanIf_ControllerModeIndication (Controller, CAN_CS_STARTED)</code> is called to inform the CanIf directly about the successful mode transition. When <code>CanIf_ControllerModeIndication (Controller, CAN_CS_STARTED)</code> returned, the request <code>Can_SetControllerMode ()</code> returns and informs CanIf about a successful request which in turn returns the upper layer request <code>CanIf_SetControllerMode ()</code> .
<b>CAN controller was in STARTED mode</b>	During the former request <code>Can_SetControllerMode ()</code> the function <code>CanIf_ControllerModeIndication (Controller, CAN_CS_STARTED)</code> is called to inform the CanIf directly about the successful mode transition (because the mode was already started). When <code>CanIf_ControllerModeIndication (Controller, CAN_CS_STARTED)</code> returned, the request <code>Can_SetControllerMode ()</code> returns and informs CanIf about a successful request which in turn returns the upper layer request <code>CanIf_SetControllerMode ()</code> .
<b>CAN controller was in SLEEP mode</b>	This transition is not allowed -> E_NOT_OK.

## 9.11 BusOff notification

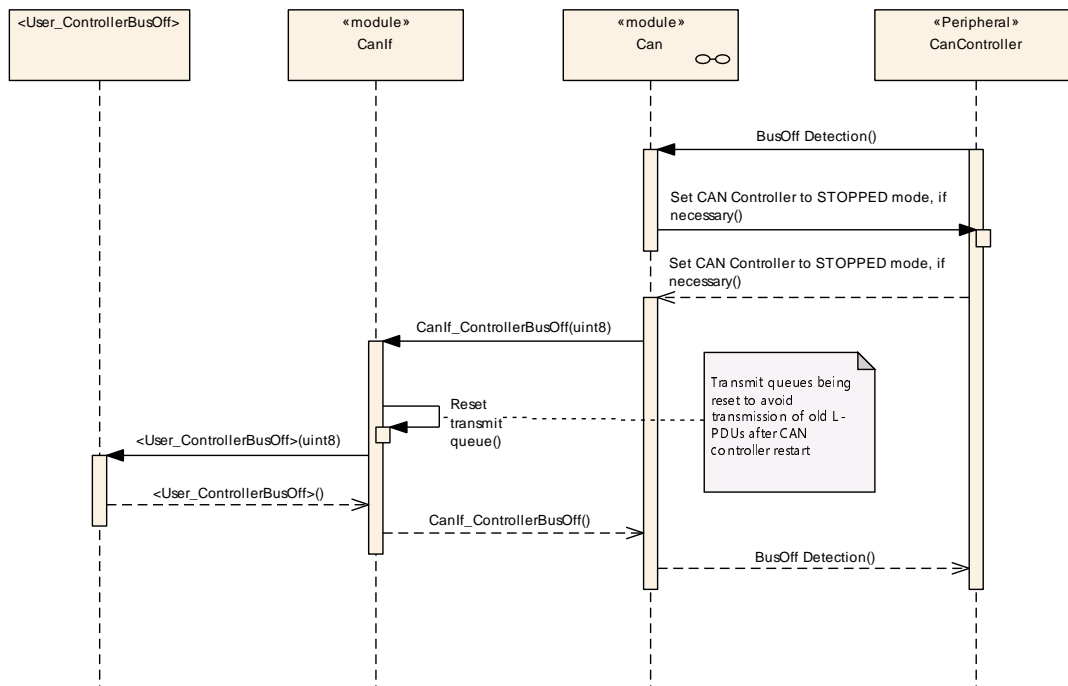


Figure 9.11: BusOff notification

Activity	Description
<b>BusOff detection interrupt</b>	The CAN controller signals a BusOff event.
<b>Stop CAN controller</b>	CAN controller is set to STOPPED mode by the CAN Driver, if necessary.
<b>BusOff indication to CAN Interface</b>	BusOff is notified to the CanIf by calling of <code>CanIf_ControllerBusOff()</code>
<b>BusOff indication to upper layer (CanSM)</b>	BusOff is notified to the upper layer by calling of <code>&lt;User_ControllerBusOff&gt;()</code>

### 9.12 BusOff recovery

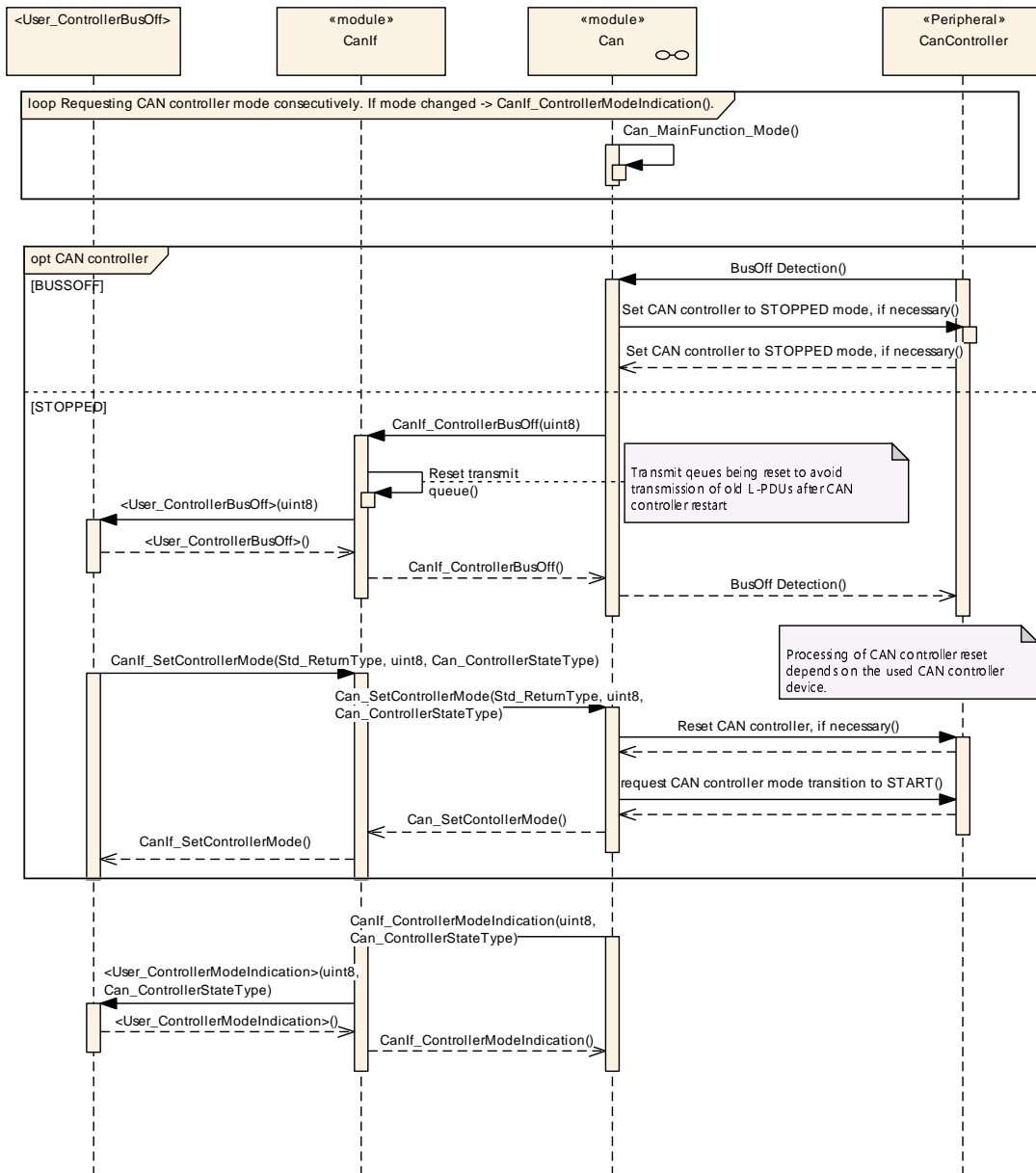


Figure 9.12: BusOff recovery

Activity	Description
<b>BusOff detection interrupt</b>	The CAN controller signals a BusOff event.
<b>Stop CAN controller</b>	CAN controller is set to STOPPED mode by the <code>CanDrv</code> , if necessary
<b>BusOff indication to <code>CanIf</code></b>	BusOff is notified to the <code>CanIf</code> by calling of <code>CanIf_ControllerBusOff()</code> . The transmit buffers inside <code>CanIf</code> will be reset.
<b>BusOff indication to upper layer</b>	BusOff is notified to the upper layer by calling of <code>&lt;User_ControllerBusOff&gt;()</code>
<b>Upper Layer (<code>CanSM</code>) initiates BusOff Recovery</b>	After a time specified by the BusOff Recovery algorithm the Recovery process itself is initiated by <code>CanIf_SetControllerMode(ControllerId, CAN_CS_STARTED)</code> .
<b>Restart of CAN controller</b>	The driver restarts the CAN controller by call of <code>Can_SetControllerMode(Controller, CAN_CS_STARTED)</code> .
<b>CAN controller started</b>	<code>CanDrv</code> informs <code>CanIf</code> about the successful start by calling <code>CanIf_ControllerModeIndication()</code> . <code>CanIf</code> 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 the [9, chapter 10.1 "Introduction to configuration specification" in SWS\_BSWGeneral].

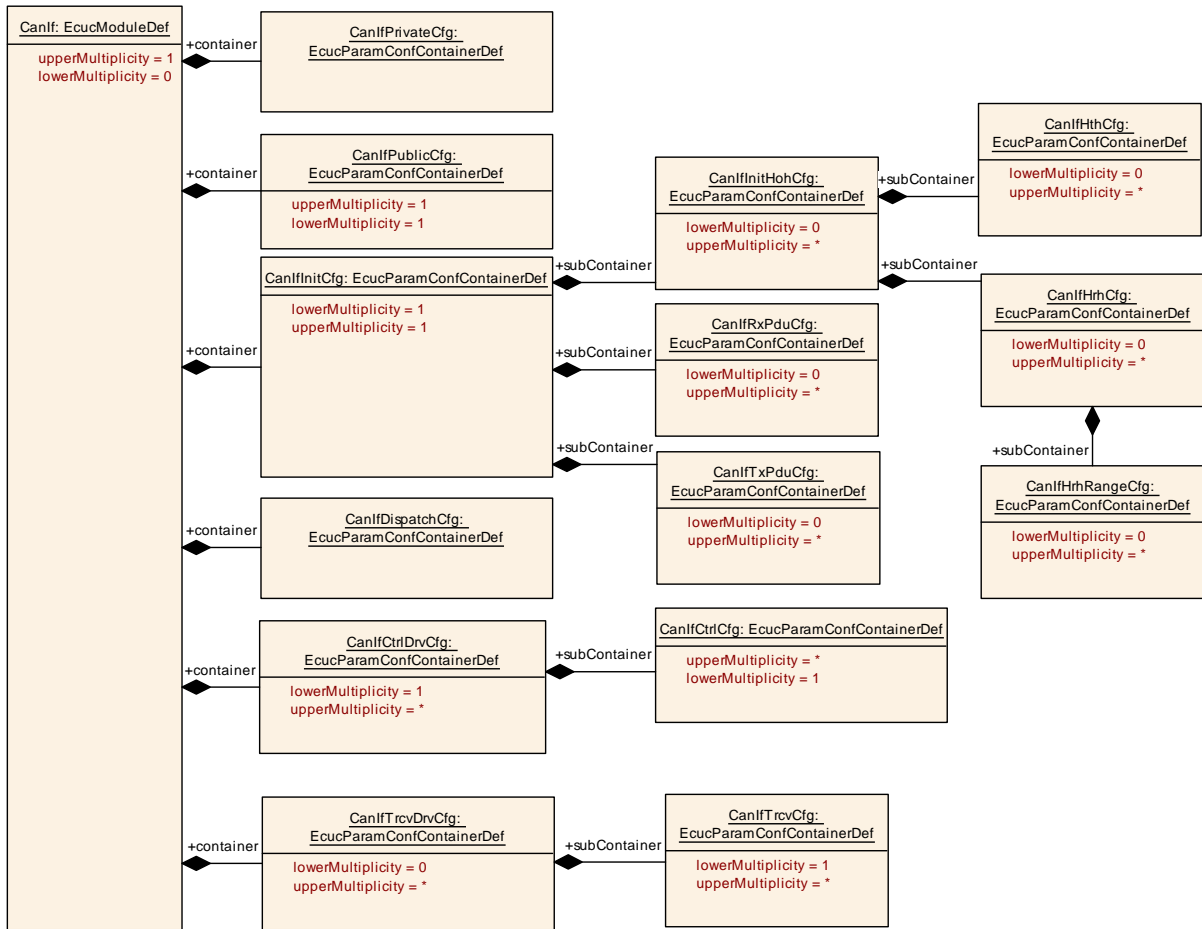
[section 10.1](#) specifies the structure (containers) and the parameters of the CanIf.

### 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

<b>SWS Item</b>	[ECUC_CanIf_00244]
<b>Module Name</b>	CanIf
<b>Description</b>	This container includes all necessary configuration sub-containers according the CAN Interface configuration structure.
<b>Post-Build Variant Support</b>	true
<b>Supported Config Variants</b>	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	Multiplicity	Scope / Dependency
<a href="#">CanIfCtrlDrvCfg</a>	1..*	Configuration parameters for all the underlying CAN Driver modules are aggregated under this container. For each CAN Driver module a separate instance of this container has to be provided.
<a href="#">CanIfDispatchCfg</a>	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.





Included Containers		
Container Name	Multiplicity	Scope / Dependency
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 separate instance of this container shall be provided.

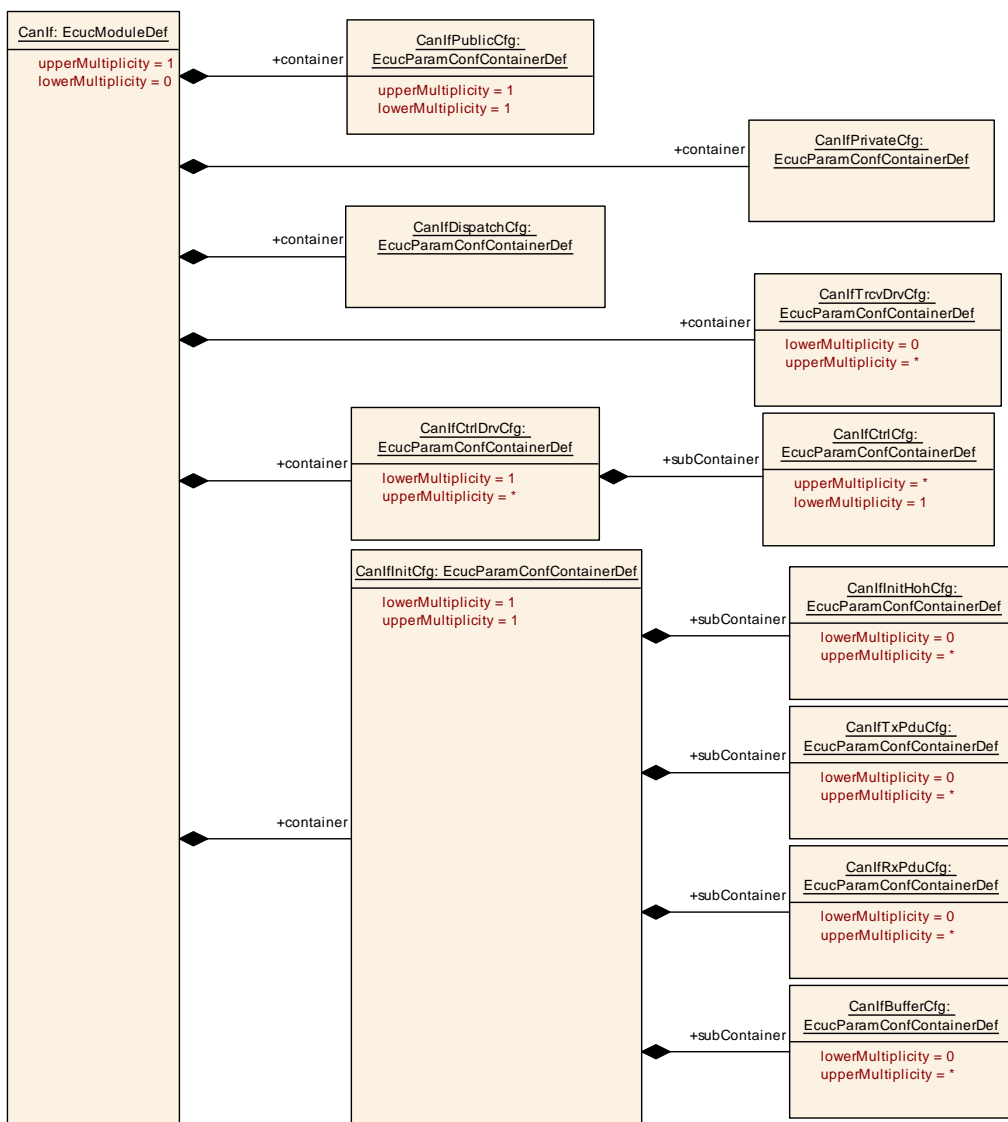


Figure 10.2: AR\_EcucDef\_CanIf

## 10.1.2 CanIfPrivateCfg

<b>SWS Item</b>	[ECUC_CanIf_00245]
<b>Container Name</b>	CanIfPrivateCfg
<b>Parent Container</b>	<a href="#">CanIf</a>
<b>Description</b>	This container contains the private configuration (parameters) of the CAN Interface.
<b>Configuration Parameters</b>	

<b>SWS Item</b>	[ECUC_CanIf_00827]		
<b>Parameter Name</b>	CanIfFixedBuffer		
<b>Parent Container</b>	<a href="#">CanIfPrivateCfg</a>		
<b>Description</b>	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).		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	[ECUC_CanIf_00617]		
<b>Parameter Name</b>	CanIfPrivateDataLengthCheck		
<b>Parent Container</b>	<a href="#">CanIfPrivateCfg</a>		
<b>Description</b>	Selects whether Data Length Check is supported. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	[ECUC_CanIf_00619]		
<b>Parameter Name</b>	CanIfPrivateSoftwareFilterType		
<b>Parent Container</b>	<a href="#">CanIfPrivateCfg</a>		







<b>Description</b>	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		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	BINARY	Selects Binary Filter method.	
	INDEX	Selects Index Filter method.	
	LINEAR	Selects Linear Filter method.	
	TABLE	Selects Table Filter method.	
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local dependency: BasicCAN reception must be enabled by referenced parameter Can HandleType of the CAN Driver module via CanIfHrhdSymRef for at least one HRH.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00675]</b>		
<b>Parameter Name</b>	CanIfSupportTTCAN		
<b>Parent Container</b>	<a href="#">CanIfPrivateCfg</a>		
<b>Description</b>	Defines whether TTCAN is supported. TRUE: TTCAN is supported. FALSE: TTCAN is not supported, only normal CAN communication is possible.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
CanIfTTGeneral	0..1	CanIfTTGeneral is specified in the SWS TTCAN Interface and defines if and in which way TTCAN is supported. This container is only included and valid if TTCAN is supported by the controller, enabled (see CanIfSupportTTCAN, ECUC_CanIf_00675), and used.

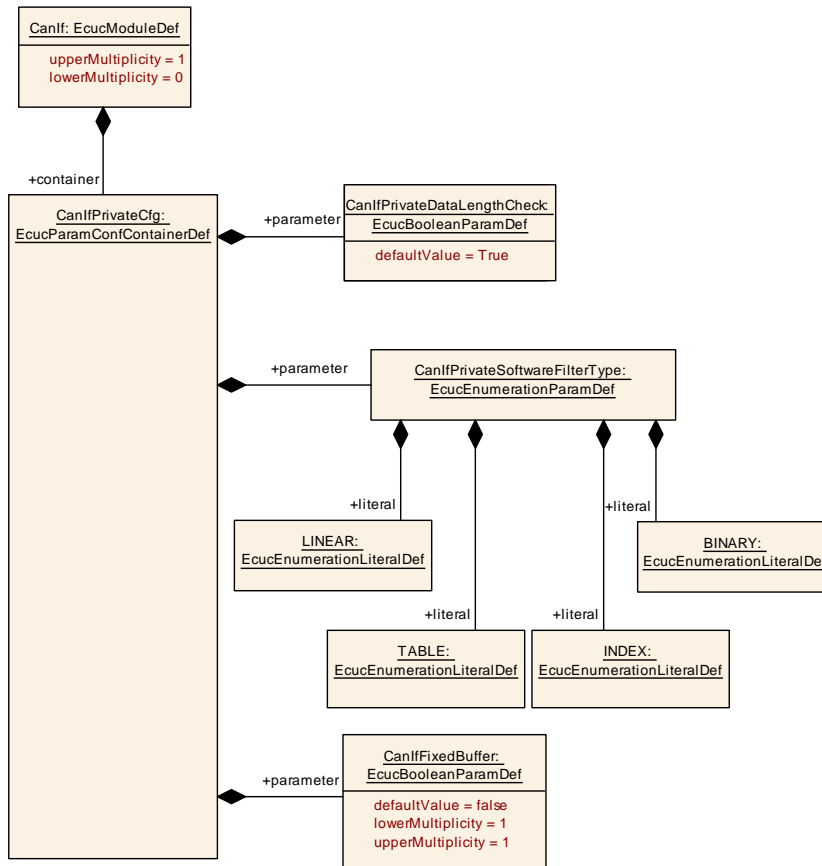


Figure 10.3: AR\_EcucDef\_CanIfPrivateCfg

### 10.1.3 CanIfPublicCfg

SWS Item	[ECUC_CanIf_00246]
Container Name	CanIfPublicCfg
Parent Container	CanIf
Description	This container contains the public configuration (parameters) of the CAN Interface.
Configuration Parameters	

SWS Item	[ECUC_CanIf_00847]		
Parameter Name	CanIfBusMirroringSupport		
Parent Container	CanIfPublicCfg		
Description	Enable support for Bus Mirroring.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	-	
	Post-build time	-	





<b>Scope / Dependency</b>	scope: local
---------------------------	--------------

<b>SWS Item</b>	<b>[ECUC_CanIf_00614]</b>		
<b>Parameter Name</b>	CanIfDevErrorDetect		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Switches the development error detection and notification on or off. <ul style="list-style-type: none"> <li>• true: detection and notification is enabled.</li> <li>• false: detection and notification is disabled.</li> </ul>		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00848]</b>		
<b>Parameter Name</b>	CanIfEnableSecurityEventReporting		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Switches the reporting of security events to the IdsM: - true: reporting is enabled. - false: reporting is disabled. <b>Tags:</b> atp.Status=draft		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00854]</b>		
<b>Parameter Name</b>	CanIfGlobalTimeSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enables/Disables the Global Time APIs used when hardware timestamping is supported. <b>Tags:</b> atp.Status=draft		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	–		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00824]</b>		
<b>Parameter Name</b>	CanIfMetaDataSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enable support for dynamic ID handling using L-SDU MetaData.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00671]</b>		
<b>Parameter Name</b>	CanIfPublicCddHeaderFile		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Defines header files for callback functions which shall be included in case of CDDs. Range of characters is 1.. 32.		
<b>Multiplicity</b>	0..*		
<b>Type</b>	EcucStringParamDef		
<b>Default value</b>	–		
<b>Length</b>	1-32		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00742]</b>		
<b>Parameter Name</b>	CanIfPublicHandleTypeEnum		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	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).		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	UINT16	–	
	UINT8	–	
<b>Post-Build Variant Value</b>	false		





<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: Can_HwHandleType		

<b>SWS Item</b>	<b>[ECUC_CanIf_00612]</b>		
<b>Parameter Name</b>	CanIfPublicMultipleDrvSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Selects support for multiple CAN Drivers. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00772]</b>		
<b>Parameter Name</b>	CanIfPublicPnSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Selects support of Partial Network features in CanIf. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00607]</b>		
<b>Parameter Name</b>	CanIfPublicReadRxPduDataApi		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enables / Disables the API CanIf_ReadRxPduData() for reading received L-SDU data. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00608]</b>		
<b>Parameter Name</b>	CanIfPublicReadRxPduNotifyStatusApi		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enables and disables the API for reading the notification status of receive L-PDUs. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00609]</b>		
<b>Parameter Name</b>	CanIfPublicReadTxPduNotifyStatusApi		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enables and disables the API for reading the notification status of transmit L-PDUs. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00610]</b>		
<b>Parameter Name</b>	CanIfPublicSetDynamicTxIdApi		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enables and disables the API for reconfiguration of the CAN Identifier for each Transmit L-PDU. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00618]</b>		
<b>Parameter Name</b>	CanIfPublicTxBuffering		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		





<b>Description</b>	Enables and disables the buffering of transmit L-PDUs (rejected by the CanDrv) within the CAN Interface module. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00733]</b>		
<b>Parameter Name</b>	CanIfPublicTxConfirmPollingSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Configuration parameter to enable/disable the API to poll for Tx Confirmation state.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	–		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local dependency: CAN State Manager module		

<b>SWS Item</b>	<b>[ECUC_CanIf_00741]</b>		
<b>Parameter Name</b>	CanIfPublicWakeupCheckValidByNM		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	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 CanIfPublicWakeupCheckValidSupport and shall only be configurable, if it is enabled. True: Enabled False: Disabled		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfPublicWakeupCheckValidSupport		

<b>SWS Item</b>	<b>[ECUC_CanIf_00611]</b>		
<b>Parameter Name</b>	CanIfPublicWakeupCheckValidSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Selects support for wake up validation True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00838]</b>		
<b>Parameter Name</b>	CanIfSetBaudrateApi		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00844]</b>		
<b>Parameter Name</b>	CanIfTriggerTransmitSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	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		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		



<b>SWS Item</b>	<b>[ECUC_CanIf_00837]</b>		
<b>Parameter Name</b>	CanIfTxOfflineActiveSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Determines whether TxOffLineActive feature (see SWS_CANIF_00072) is supported by CanIf. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00613]</b>		
<b>Parameter Name</b>	CanIfVersionInfoApi		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Enables and disables the API for reading the version information about the CAN Interface. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00843]</b>		
<b>Parameter Name</b>	CanIfWakeupSupport		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	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		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
<a href="#">CanIfSecurityEventRefs</a>	0..1	<p>Container for the references to IdsMEvent elements representing the security events that the CanIf module shall report to the IdsM in case the corresponding 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.</p> <p><b>Tags:</b> atp.Status=draft</p>

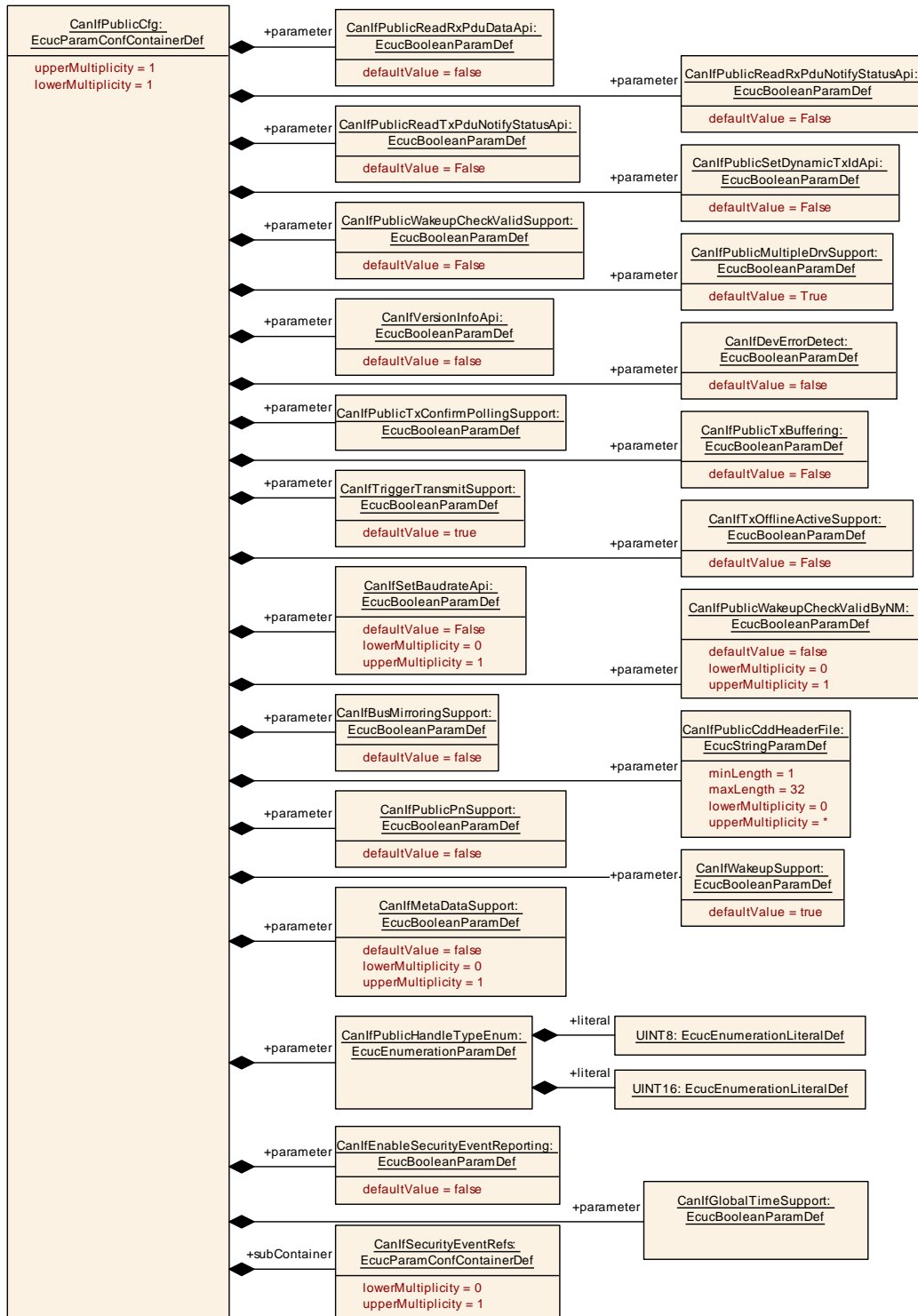


Figure 10.4: AR\_EcucDef\_CanIfPublicCfg

### 10.1.4 CanIfInitCfg

<b>SWS Item</b>	[ECUC_CanIf_00247]
<b>Container Name</b>	CanIfInitCfg
<b>Parent Container</b>	CanIf
<b>Description</b>	This container contains the init parameters of the CAN Interface.
<b>Configuration Parameters</b>	

<b>SWS Item</b>	[ECUC_CanIf_00623]		
<b>Parameter Name</b>	CanIfInitCfgSet		
<b>Parent Container</b>	CanIfInitCfg		
<b>Description</b>	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 Drivers. constant to CanIf_ConfigType		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucStringParamDef		
<b>Default value</b>	-		
<b>Length</b>	1-32		
<b>Regular Expression</b>	-		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	[ECUC_CanIf_00828]		
<b>Parameter Name</b>	CanIfMaxBufferSize		
<b>Parent Container</b>	CanIfInitCfg		
<b>Description</b>	Maximum total size of all Tx buffers. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 18446744073709551615		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00830]</b>		
<b>Parameter Name</b>	CanIfMaxRxPduCfg		
<b>Parent Container</b>	<a href="#">CanIfInitCfg</a>		
<b>Description</b>	Maximum number of Pdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 18446744073709551615		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00829]</b>		
<b>Parameter Name</b>	CanIfMaxTxPduCfg		
<b>Parent Container</b>	<a href="#">CanIfInitCfg</a>		
<b>Description</b>	Maximum number of Pdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 18446744073709551615		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: local		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
<a href="#">CanIfBufferCfg</a>	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.





Included Containers		
Container Name	Multiplicity	Scope / Dependency
CanIfInitHohCfg	0..*	This container contains the references to the configuration setup of each underlying CAN Driver.
CanIfRxPduCfg	0..*	<p>This container contains the configuration (parameters) of each receive CAN L-PDU.</p> <p>The SHORT-NAME of "CanIfRxPduConfig" container itself represents the symbolic name of Receive L-PDU.</p> <p>This L-SDU produces a meta data item of type CAN_ID_32, and, depending on CanIfRxPduXLParams, meta data items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32.</p>
CanIfTxPduCfg	0..*	<p>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.</p> <p>The SHORT-NAME of "CanIfTxPduConfig" container represents the symbolic name of Transmit L-PDU.</p> <p>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.</p>

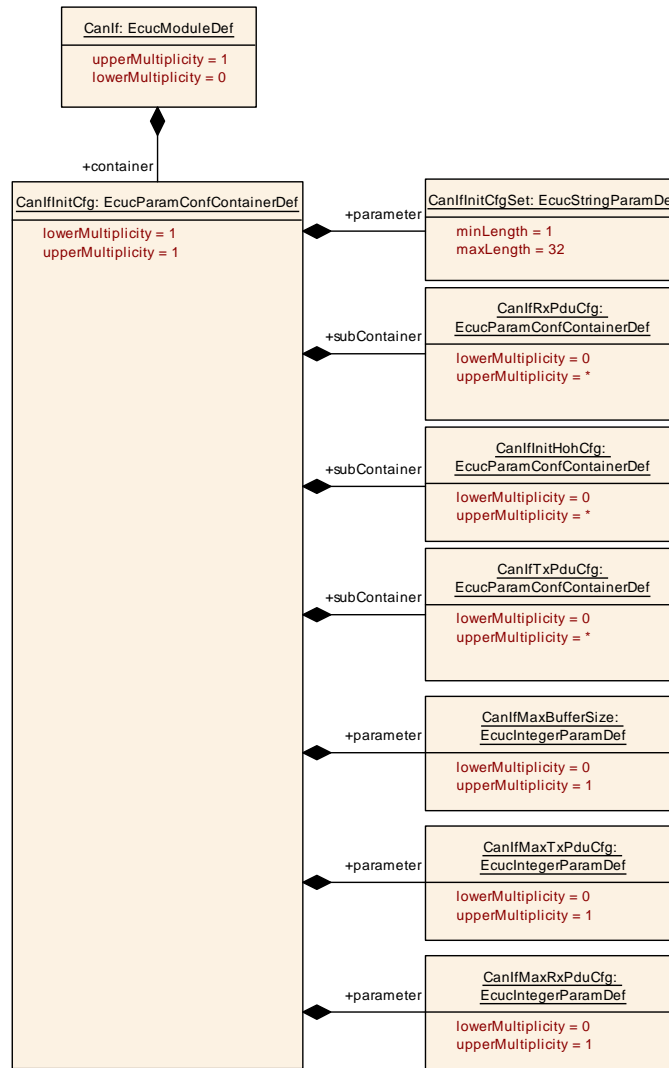


Figure 10.5: AR\_EcucDef\_CanIfInitCfg

### 10.1.5 CanIfTxPduCfg

<b>SWS Item</b>	[ECUC_CanIf_00248]		
<b>Container Name</b>	CanIfTxPduCfg		
<b>Parent Container</b>	<a href="#">CanIfInitCfg</a>		
<b>Description</b>	<p>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.</p> <p>The SHORT-NAME of "CanIfTxPduConfig" container represents the symbolic name of Transmit L-PDU.</p> <p>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.</p>		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Multiplicity Configuration Class</b>	Pre-compile time	X	VARIANT-PRE-COMPILE





	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Configuration Parameters</b>			

<b>SWS Item</b>	<b>[ECUC_CanIf_00592]</b>		
<b>Parameter Name</b>	CanIfTxPduCanId		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	CAN Identifier of transmit CAN L-PDUs used by the CAN Driver for CAN L-PDU transmission. Range: 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.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU  dependency: Not relevant if CanIfTxPduXLParams is configured and CanIfXLSduType is not 3.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00823]</b>		
<b>Parameter Name</b>	CanIfTxPduCanIdMask		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	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: 11 bits for Standard CAN Identifier, 29 bits for Extended CAN Identifier.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 3758096383		
<b>Default value</b>	3758096383		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		



<b>SWS Item</b>	<b>[ECUC_CanIf_00590]</b>		
<b>Parameter Name</b>	CanIfTxPduCanIdType		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	Type of CAN Identifier of the transmit CAN L-PDU used by the CAN Driver module for CAN L-PDU transmission.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	EXTENDED_CAN		CAN frame with extended identifier (29 bits)
	EXTENDED_FD_CAN		CAN FD frame with extended identifier (29 bits)
	STANDARD_CAN		CAN frame with standard identifier (11 bits)
	STANDARD_FD_CAN		CAN FD frame with standard identifier (11 bits)
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU dependency: Not relevant if CanIfTxPduXLParams is configured and CanIfXLSduType is not 3, otherwise mandatory.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00591]</b>		
<b>Parameter Name</b>	CanIfTxPduId		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	ECU wide unique, symbolic handle for transmit CAN L-SDU. Range: 0..max. number of CanIfTxPduIds		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
<b>Range</b>	0 .. 4294967295		
<b>Default value</b>	-		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	-	
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00773]</b>		
<b>Parameter Name</b>	CanIfTxPduPnFilterPdu		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		





<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: This parameter shall only be configurable if CanIfPublicPnSupport equals True.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00589]</b>		
<b>Parameter Name</b>	CanIfTxPduReadNotifyStatus		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	Enables and disables transmit confirmation for each transmit CAN L-SDU for reading its notification status. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: CanIfPublicReadTxPduNotifyStatusApi must be enabled.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00840]</b>		
<b>Parameter Name</b>	CanIfTxPduTriggerTransmit		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	Determines if or if not CanIf shall use the trigger transmit API for this PDU.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU dependency: If CanIfTxPduTriggerTransmit is TRUE then CanIfTxPduUserTx ConfirmationUL has to be either PDUR or CDD and CanIfTxPduUserTriggerTransmit Name has to be specified accordingly.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00845]</b>		
<b>Parameter Name</b>	CanIfTxPduTruncation		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		





<b>Description</b>	Enables/disables truncation of PDUs that exceed the configured size.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00593]</b>		
<b>Parameter Name</b>	CanIfTxPduType		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	Defines the type of each transmit CAN L-PDU.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	DYNAMIC	CAN ID is defined at runtime.	
	STATIC	CAN ID is defined at compile-time.	
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00842]</b>		
<b>Parameter Name</b>	CanIfTxPduUserTriggerTransmitName		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	<p>This parameter defines the name of the &lt;User_TriggerTransmit&gt;. This parameter depends on the parameter CanIfTxPduUserTxConfirmationUL. If CanIfTxPduUserTxConfirmationUL equals CAN_TP, CAN_NM, PDUR, XCP, CAN_TSYN, J1939NM or J1939TP, the name of the &lt;User_TriggerTransmit&gt; is fixed. If CanIfTxPduUserTxConfirmationUL equals CDD, the name of the &lt;User_TxConfirmation&gt; is selectable.</p> <p>Please be aware that this parameter depends on the same parameter as CanIfTxPduUserTxConfirmationName. It shall be clear which upper layer is responsible for that PDU.</p>		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	-		
<b>Length</b>	1-32		
<b>Regular Expression</b>	-		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE





	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfTxPduUserTriggerTransmitName requires CanIfTxPduUserTxConfirmationUL to be either PDUR or CDD.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00528]</b>		
<b>Parameter Name</b>	CanIfTxPduUserTxConfirmationName		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	This parameter defines the name of the <User_TxConfirmation>. This parameter depends on the parameter CanIfTxPduUserTxConfirmationUL. If CanIfTxPduUserTxConfirmationUL equals CAN_TP, CAN_NM, PDUR, XCP, CAN_TSYN, J1939NM or J1939TP, the name of the <User_TxConfirmation> is fixed. If CanIfTxPduUserTxConfirmationUL equals CDD, the name of the <User_TxConfirmation> is selectable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Length</b>	1-32		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00527]</b>		
<b>Parameter Name</b>	CanIfTxPduUserTxConfirmationUL		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	This parameter defines the upper layer (UL) module to which the confirmation of the successfully transmitted CanTxPduId has to be routed via the <User_TxConfirmation>. This <User_TxConfirmation> has to be invoked when the confirmation of the configured CanTxPduId will be received by a Tx confirmation event from the CAN Driver module. If no upper layer (UL) module is configured, no <User_TxConfirmation> has to be called in case of a Tx confirmation event of the CanTxPduId from the CAN Driver module.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_NM	CAN NM	
	CAN_TP	CAN TP	
	CAN_TSYN	Global Time Synchronization over CAN	
	CDD	Complex Driver	
	J1939NM	J1939Nm	
	J1939TP	J1939Tp	
	PDUR	PDU Router	





	XCP	Extended Calibration Protocol	
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00831]</b>		
<b>Parameter Name</b>	CanIfTxPduBufferRef		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	Configurable reference to a CanIf buffer configuration.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to <a href="#">CanIfBufferCfg</a>		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00603]</b>		
<b>Parameter Name</b>	CanIfTxPduRef		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to Pdu		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
<a href="#">CanIfTTTxFrameTriggering</a>	0..1	CanIfTTTxFrameTriggering is specified in the SWS TTCAN Interface and defines Frame trigger for TTCAN transmission.  This container is only included and valid if TTCAN is supported by the controller, enabled (see CanIfSupportTTCAN, ECUC_CanIf_00675), and a joblist is used.
<a href="#">CanIfTxPduXLParams</a>	0..1	CAN XL parameters. Identifies the CanIfTxPduCfg as CAN XL PDU.

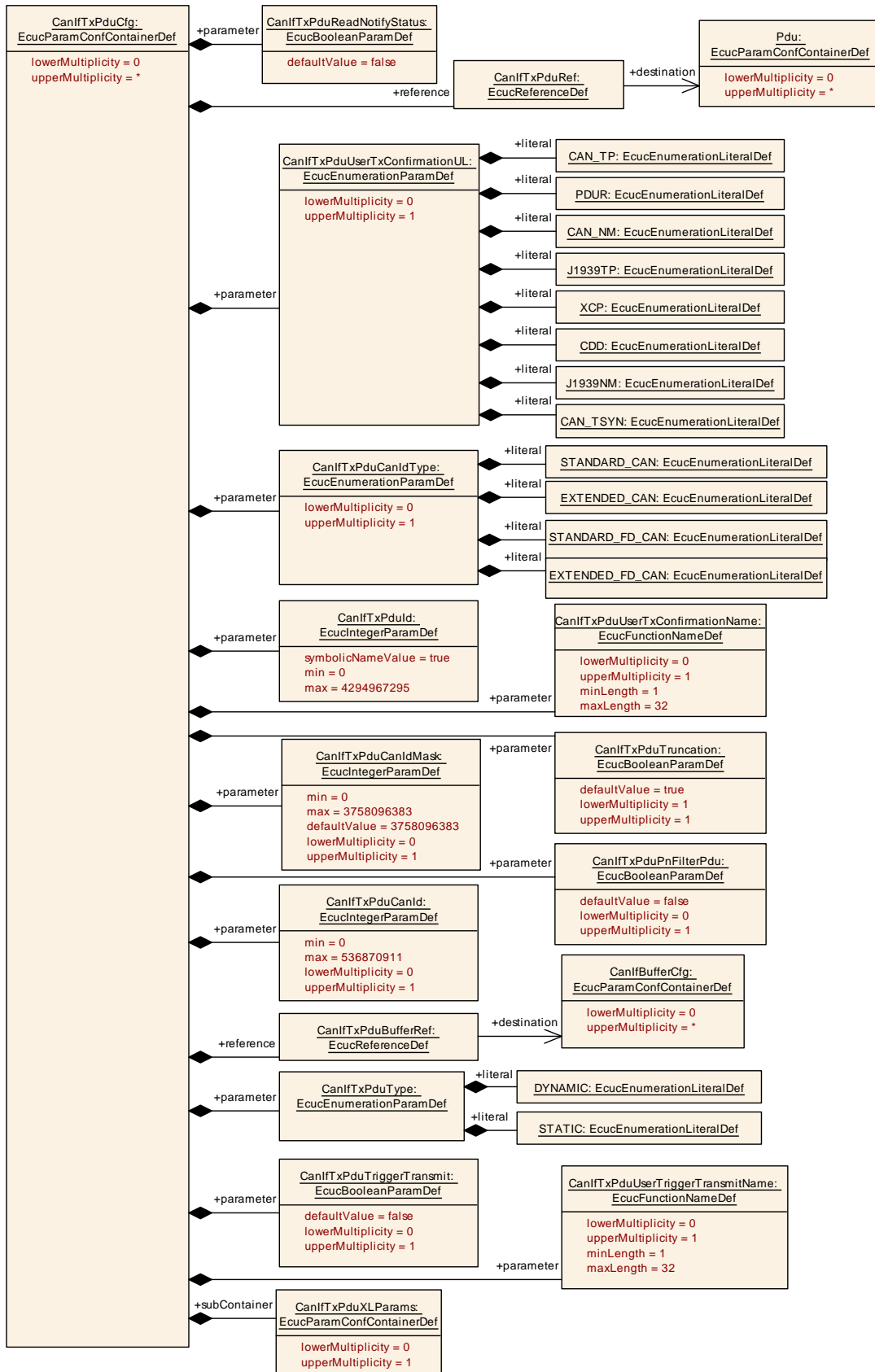


Figure 10.6: AR\_EcucDef\_CanIfTxPduCfg

### 10.1.6 CanIfTxPduXLParams

<b>SWS Item</b>	[ECUC_CanIf_00855]		
<b>Container Name</b>	CanIfTxPduXLParams		
<b>Parent Container</b>	<a href="#">CanIfTxPduCfg</a>		
<b>Description</b>	CAN XL parameters. Identifies the CanIfTxPduCfg as CAN XL PDU.		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Configuration Parameters</b>			

<b>SWS Item</b>	[ECUC_CanIf_00859]		
<b>Parameter Name</b>	CanIfXLAcceptanceField		
<b>Parent Container</b>	<a href="#">CanIfTxPduXLParams</a>		
<b>Description</b>	Acceptance field of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 4294967295		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: Not relevant for CanIfXLSduType = 3.		

<b>SWS Item</b>	[ECUC_CanIf_00856]		
<b>Parameter Name</b>	CanIfXLPriorityId		
<b>Parent Container</b>	<a href="#">CanIfTxPduXLParams</a>		
<b>Description</b>	Priority ID of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 2047		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD





<b>Scope / Dependency</b>	scope: local
---------------------------	--------------

<b>SWS Item</b>	<b>[ECUC_CanIf_00858]</b>		
<b>Parameter Name</b>	CanIfXLSduType		
<b>Parent Container</b>	<a href="#">CanIfTxPduXLParams</a>		
<b>Description</b>	SDU type of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 255		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00857]</b>		
<b>Parameter Name</b>	CanIfXLVcid		
<b>Parent Container</b>	<a href="#">CanIfTxPduXLParams</a>		
<b>Description</b>	Virtual CAN network ID of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 255		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>No Included Containers</b>
-------------------------------



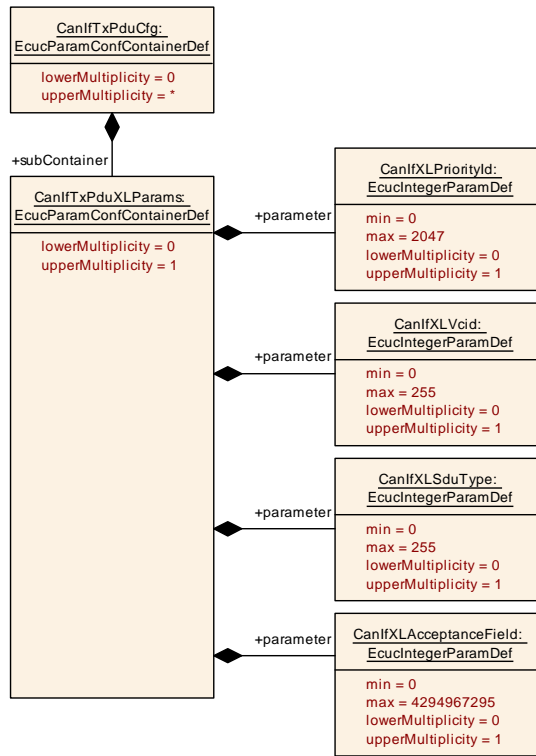


Figure 10.7: AR\_EcucDef\_CanIfTxPduXLParams

### 10.1.7 CanIfRxPduCfg

<b>SWS Item</b>	[ECUC_CanIf_00249]		
<b>Container Name</b>	CanIfRxPduCfg		
<b>Parent Container</b>	<a href="#">CanIfInitCfg</a>		
<b>Description</b>	<p>This container contains the configuration (parameters) of each receive CAN L-PDU.</p> <p>The SHORT-NAME of "CanIfRxPduConfig" container itself represents the symbolic name of Receive L-PDU.</p> <p>This L-SDU produces a meta data item of type CAN_ID_32, and, depending on CanIfRxPduXLParams, meta data items of type PRIORITYID_16, VLAN_16, SDUTYPE_8, and ACCEPTANCEFIELD_32.</p>		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Configuration Parameters</b>			

<b>SWS Item</b>	[ECUC_CanIf_00598]		
<b>Parameter Name</b>	CanIfRxPduCanId		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		





<b>Description</b>	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: 11 Bit For Standard CAN Identifier ... 29 Bit For Extended CAN identifier		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	[ECUC_CanIf_00822]		
<b>Parameter Name</b>	CanIfRxPduCanIdMask		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	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: 11 bits for Standard CAN Identifier, 29 bits for Extended CAN Identifier.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	536870911		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	[ECUC_CanIf_00596]		
<b>Parameter Name</b>	CanIfRxPduCanIdType		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	CAN Identifier of receive CAN L-PDUs used by the CAN Driver for CAN L-PDU reception.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	EXTENDED_CAN		CAN 2.0 or CAN FD frame with extended identifier (29 bits)





	EXTENDED_FD_CAN	CAN FD frame with extended identifier (29 bits)	
	EXTENDED_NO_FD_CAN	CAN 2.0 frame with extended identifier (29 bits)	
	STANDARD_CAN	CAN 2.0 or CAN FD frame with standard identifier (11 bits)	
	STANDARD_FD_CAN	CAN FD frame with standard identifier (11 bits)	
	STANDARD_NO_FD_CAN	CAN 2.0 frame with standard identifier (11 bits)	
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local  dependency: Not relevant if CanIfRxPduXLParams is configured and CanIfXLSduType is not 3, otherwise mandatory.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00599]</b>		
<b>Parameter Name</b>	CanIfRxPduDataLength		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 2048		
<b>Default value</b>	-		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU  dependency: If CanIfRxPduDataLength > 8 then CanIfRxPduCanIdType must not be STANDARD_NO_FD_CAN or EXTENDED_NO_FD_CAN If CanIfRxPduDataLength > 64 then CanIfRxPduXLParams must be present.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00846]</b>		
<b>Parameter Name</b>	CanIfRxPduDataLengthCheck		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Value</b>	true		





<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	[ECUC_CanIf_00597]		
<b>Parameter Name</b>	CanIfRxPduId		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	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		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
<b>Range</b>	0 .. 4294967295		
<b>Default value</b>	-		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	-	
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: ECU dependency: Not relevant if CanIfRxPduXLParams is configured and CanIfXLSduType is not 3.		

<b>SWS Item</b>	[ECUC_CanIf_00600]		
<b>Parameter Name</b>	CanIfRxPduReadData		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	Enables and disables the Rx buffering for reading of received L-SDU data. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	[ECUC_CanIf_00595]		
<b>Parameter Name</b>	CanIfRxPduReadNotifyStatus		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	Enables and disables receive indication for each receive CAN L-SDU for reading its notification status. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	true		





<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: CanIfPublicReadRxPduNotifyStatusApi must be enabled.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00530]</b>		
<b>Parameter Name</b>	CanIfRxPduUserRxIndicationName		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	This parameter defines the name of the <User_RxIndication>. This parameter depends on the parameter CanIfRxPduUserRxIndicationUL. If CanIfRxPduUserRxIndicationUL equals CAN_TP, CAN_NM, PDUR, XCP, CAN_TSYN, J1939NM or J1939TP, the name of the <User_RxIndication> is fixed. If CanIfRxPduUserRxIndicationUL equals CDD, the name of the <User_RxIndication> is selectable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	-		
<b>Length</b>	1-32		
<b>Regular Expression</b>	-		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00529]</b>		
<b>Parameter Name</b>	CanIfRxPduUserRxIndicationUL		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	This parameter defines the upper layer (UL) module to which the indication of the successfully received CANRXPDUID has to be routed via <User_RxIndication>. This <User_RxIndication> has to be invoked when the indication of the configured CANRXPDUID will be received by an Rx indication event from the CAN Driver module. If no upper layer (UL) module is configured, no <User_RxIndication> has to be called in case of an Rx indication event of the CANRXPDUID from the CAN Driver module.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_NM	CAN NM	
	CAN_TP	CAN TP	
	CAN_TSYN	Global Time Synchronization over CAN	
	CDD	Complex Driver	
	J1939NM	J1939Nm	
	J1939TP	J1939Tp	
	PDUR	PDU Router	
	XCP	Extended Calibration Protocol	





<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00602]</b>		
<b>Parameter Name</b>	CanIfRxPduHrhdRef		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	The HRH to which Rx L-PDU belongs to, is referred through this parameter.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to <a href="#">CanIfHrhCfg</a>		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: This information has to be derived from the CAN Driver configuration.		

<b>SWS Item</b>	<b>[ECUC_CanIf_00601]</b>		
<b>Parameter Name</b>	CanIfRxPduRef		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to Pdu		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>Included Containers</b>		
<b>Container Name</b>	<b>Multiplicity</b>	<b>Scope / Dependency</b>
<a href="#">CanIfRxPduCanIdRange</a>	0..1	Optional container that allows to map a range of CAN Ids to one PduId.
<a href="#">CanIfRxPduXLParams</a>	0..1	CAN XL parameters. Identifies the CanIfRxPduCfg as CAN XL PDU.
CanIfTTRxFrameTriggering	0..1	CanIfTTRxFrameTriggering is specified in the SWS TTCAN Interface and defines Frame trigger for TTCAN reception.  This container is only included and valid if TTCAN is supported by the controller, enabled (see CanIfSupportTTCAN, ECUC_CanIf_00675), and a joblist is used for reception.

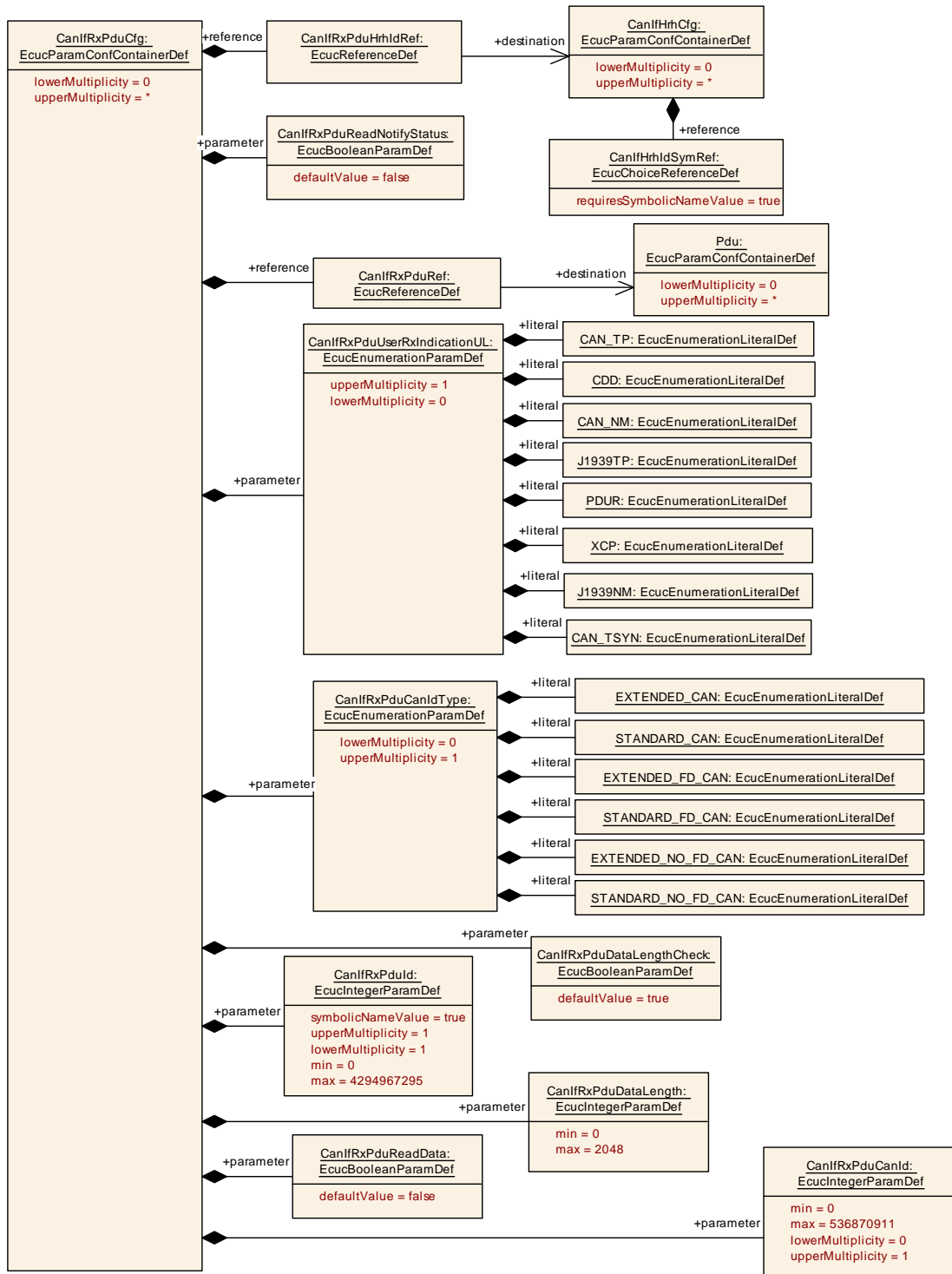


Figure 10.8: AR\_EcucDef\_CanIfRxPduCfg

### 10.1.8 CanIfRxPduXLParams

<b>SWS Item</b>	[ECUC_CanIf_00860]		
<b>Container Name</b>	CanIfRxPduXLParams		
<b>Parent Container</b>	<a href="#">CanIfRxPduCfg</a>		
<b>Description</b>	CAN XL parameters. Identifies the CanIfRxPduCfg as CAN XL PDU.		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Configuration Parameters</b>			

<b>SWS Item</b>	[ECUC_CanIf_00864]		
<b>Parameter Name</b>	CanIfXLAcceptanceField		
<b>Parent Container</b>	<a href="#">CanIfRxPduXLParams</a>		
<b>Description</b>	Acceptance field of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 4294967295		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: Not relevant for CanIfXLSduType = 3.		

<b>SWS Item</b>	[ECUC_CanIf_00861]		
<b>Parameter Name</b>	CanIfXLPriorityId		
<b>Parent Container</b>	<a href="#">CanIfRxPduXLParams</a>		
<b>Description</b>	Priority ID of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 2047		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD







<b>Scope / Dependency</b>	scope: local
---------------------------	--------------

<b>SWS Item</b>	<b>[ECUC_CanIf_00863]</b>		
<b>Parameter Name</b>	CanIfXLSduType		
<b>Parent Container</b>	<a href="#">CanIfRxPduXLParams</a>		
<b>Description</b>	SDU type of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 255		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00862]</b>		
<b>Parameter Name</b>	CanIfXLVcid		
<b>Parent Container</b>	<a href="#">CanIfRxPduXLParams</a>		
<b>Description</b>	Virtual CAN network ID of a CAN XL message.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 255		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>No Included Containers</b>
-------------------------------

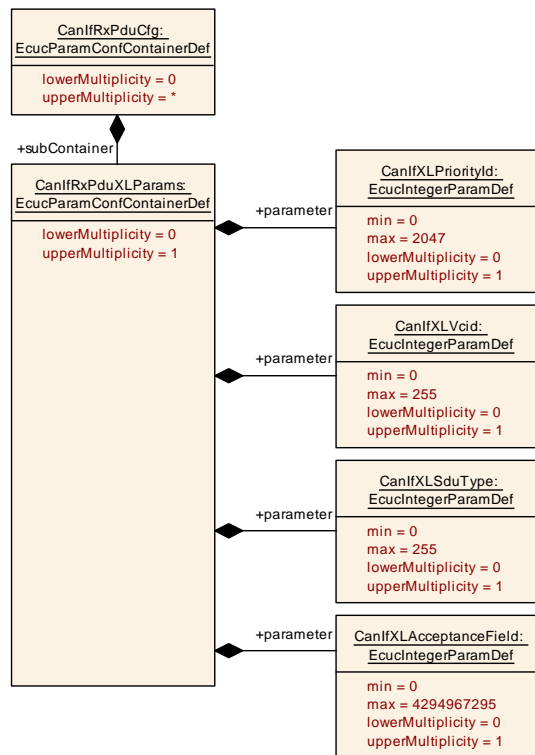


Figure 10.9: AR\_EcucDef\_CanIfRxPduXLParams

### 10.1.9 CanIfRxPduCanIdRange

SWS Item	[ECUC_CanIf_00743]
Container Name	CanIfRxPduCanIdRange
Parent Container	<a href="#">CanIfRxPduCfg</a>
Description	Optional container that allows to map a range of CAN Ids to one Pdul.
Configuration Parameters	

SWS Item	[ECUC_CanIf_00745]		
Parameter Name	CanIfRxPduCanIdRangeLowerCanId		
Parent Container	<a href="#">CanIfRxPduCanIdRange</a>		
Description	Lower CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids are mapped to one Pdul.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 536870911		
Default value	–		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00744]</b>		
<b>Parameter Name</b>	CanIfRxPduCanIdRangeUpperCanId		
<b>Parent Container</b>	<a href="#">CanIfRxPduCanIdRange</a>		
<b>Description</b>	Upper CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids are mapped to one Pdul.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	-		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

No Included Containers

### 10.1.10 CanIfDispatchCfg

<b>SWS Item</b>	<b>[ECUC_CanIf_00250]</b>		
<b>Container Name</b>	CanIfDispatchCfg		
<b>Parent Container</b>	<a href="#">CanIf</a>		
<b>Description</b>	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.		
<b>Configuration Parameters</b>			

<b>SWS Item</b>	<b>[ECUC_CanIf_00791]</b>		
<b>Parameter Name</b>	CanIfDispatchUserCheckTrcvWakeFlagIndicationName		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the name of <User_CheckTrcvWakeFlagIndication>. If CanIfDispatchUserCheckTrcvWakeFlagIndicationUL equals CAN_SM the name of <User_CheckTrcvWakeFlagIndication> is fixed. If it equals CDD, the name is selectable. If CanIfPublicPnSupport equals False, this parameter shall not be configurable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	-		
<b>Regular Expression</b>	-		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD





	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserCheckTrcvWakeFlagIndicationUL, CanIfPublicPnSupport		

<b>SWS Item</b>	<b>[ECUC_CanIf_00792]</b>		
<b>Parameter Name</b>	CanIfDispatchUserCheckTrcvWakeFlagIndicationUL		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the upper layer module to which the CheckTrcvWakeFlag Indication from the Driver modules have to be routed. If CanIfPublicPnSupport equals False, this parameter shall not be configurable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_SM	CAN State Manager	
	CDD	Complex Driver	
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfPublicPnSupport		

<b>SWS Item</b>	<b>[ECUC_CanIf_00789]</b>		
<b>Parameter Name</b>	CanIfDispatchUserClearTrcvWufFlagIndicationName		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the name of <User_ClearTrcvWufFlagIndication>. If CanIfDispatchUserClearTrcvWufFlagIndicationUL equals CAN_SM the name of <User_ClearTrcvWufFlagIndication> is fixed. If it equals CDD, the name is selectable. If CanIfPublicPnSupport equals False, this parameter shall not be configurable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	





<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserClearTrcvWufFlagIndicationUL, CanIfPublicPnSupport		
<b>SWS Item</b>	[ECUC_CanIf_00790]		
<b>Parameter Name</b>	CanIfDispatchUserClearTrcvWufFlagIndicationUL		
<b>Parent Container</b>	CanIfDispatchCfg		
<b>Description</b>	This parameter defines the upper layer module to which the ClearTrcvWufFlag Indication from the Driver modules have to be routed. If CanIfPublicPnSupport equals False, this parameter shall not be configurable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_SM	CAN State Manager	
	CDD	Complex Driver	
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfPublicPnSupport		

<b>SWS Item</b>	[ECUC_CanIf_00819]		
<b>Parameter Name</b>	CanIfDispatchUserConfirmPnAvailabilityName		
<b>Parent Container</b>	CanIfDispatchCfg		
<b>Description</b>	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 False, this parameter shall not be configurable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	





<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserConfirmPnAvailabilityUL, CanIfPublicPnSupport
---------------------------	--

<b>SWS Item</b>	<b>[ECUC_CanIf_00820]</b>		
<b>Parameter Name</b>	CanIfDispatchUserConfirmPnAvailabilityUL		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the upper layer module to which the ConfirmPnAvailability notification from the Driver modules have to be routed. If CanIfPublicPnSupport equals False, this parameter shall not be configurable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_SM	CAN State Manager	
	CDD	Complex Driver	
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfPublicPnSupport		

<b>SWS Item</b>	<b>[ECUC_CanIf_00525]</b>		
<b>Parameter Name</b>	CanIfDispatchUserCtrlBusOffName		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Length</b>	1-32		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	





<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserCtrlBusOffUL
---------------------------	---

<b>SWS Item</b>	<b>[ECUC_CanIf_00547]</b>		
<b>Parameter Name</b>	CanIfDispatchUserCtrlBusOffUL		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	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>.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_SM	CAN State Manager	
	CDD	Complex Driver	
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00683]</b>		
<b>Parameter Name</b>	CanIfDispatchUserCtrlModelIndicationName		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the name of <User_ControllerModelIndication>. This parameter depends on the parameter CanIfDispatchUserCtrlModelIndicationUL. If CanIfDispatchUserCtrlModelIndicationUL equals CAN_SM the name of <User_ControllerModelIndication> is fixed. If CanIfDispatchUserCtrlModelIndicationUL equals CDD, the name of <User_ControllerModelIndication> is selectable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Length</b>	1-32		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserCtrlModelIndicationUL		

<b>SWS Item</b>	<b>[ECUC_CanIf_00684]</b>		
<b>Parameter Name</b>	CanIfDispatchUserCtrlModelIndicationUL		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	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_ControllerModelIndication>.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_SM	CAN State Manager	
	CDD	Complex Driver	
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00685]</b>		
<b>Parameter Name</b>	CanIfDispatchUserTrcvModelIndicationName		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the name of <User_TrvcModelIndication>. This parameter depends on the parameter CanIfDispatchUserTrcvModelIndicationUL. If CanIfDispatchUserTrcvModelIndicationUL equals CAN_SM the name of <User_TrvcModelIndication> is fixed. If CanIfDispatchUserTrcvModelIndicationUL equals CDD, the name of <User_TrvcModelIndication> is selectable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Length</b>	1-32		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserTrcvModelIndicationUL		

<b>SWS Item</b>	<b>[ECUC_CanIf_00686]</b>		
<b>Parameter Name</b>	CanIfDispatchUserTrcvModelIndicationUL		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	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_TrvcModelIndication>. If no UL module is configured, no upper layer callback function will be called.		







<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CAN_SM	CAN State Manager	
	CDD	Complex Driver	
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00531]</b>		
<b>Parameter Name</b>	CanIfDispatchUserValidateWakeupEventName		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	This parameter defines the name of <User_ValidateWakeupEvent>. This parameter depends on the parameter CanIfDispatchUserValidateWakeupEventUL. If CanIfDispatchUserValidateWakeupEventUL equals ECUM, the name of <User_ValidateWakeupEvent> is fixed. If CanIfDispatchUserValidateWakeupEventUL equals CDD, the name of <User_ValidateWakeupEvent> is selectable.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucFunctionNameDef		
<b>Default value</b>	–		
<b>Length</b>	1-32		
<b>Regular Expression</b>	–		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: CanIfDispatchUserValidateWakeupEventUL		

<b>SWS Item</b>	<b>[ECUC_CanIf_00549]</b>		
<b>Parameter Name</b>	CanIfDispatchUserValidateWakeupEventUL		
<b>Parent Container</b>	<a href="#">CanIfDispatchCfg</a>		
<b>Description</b>	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_ValidateWakeupEvent>.		
<b>Multiplicity</b>	0..1		





<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	CDD	Complex Driver	
	ECUM	ECU State Manager	
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>No Included Containers</b>
-------------------------------

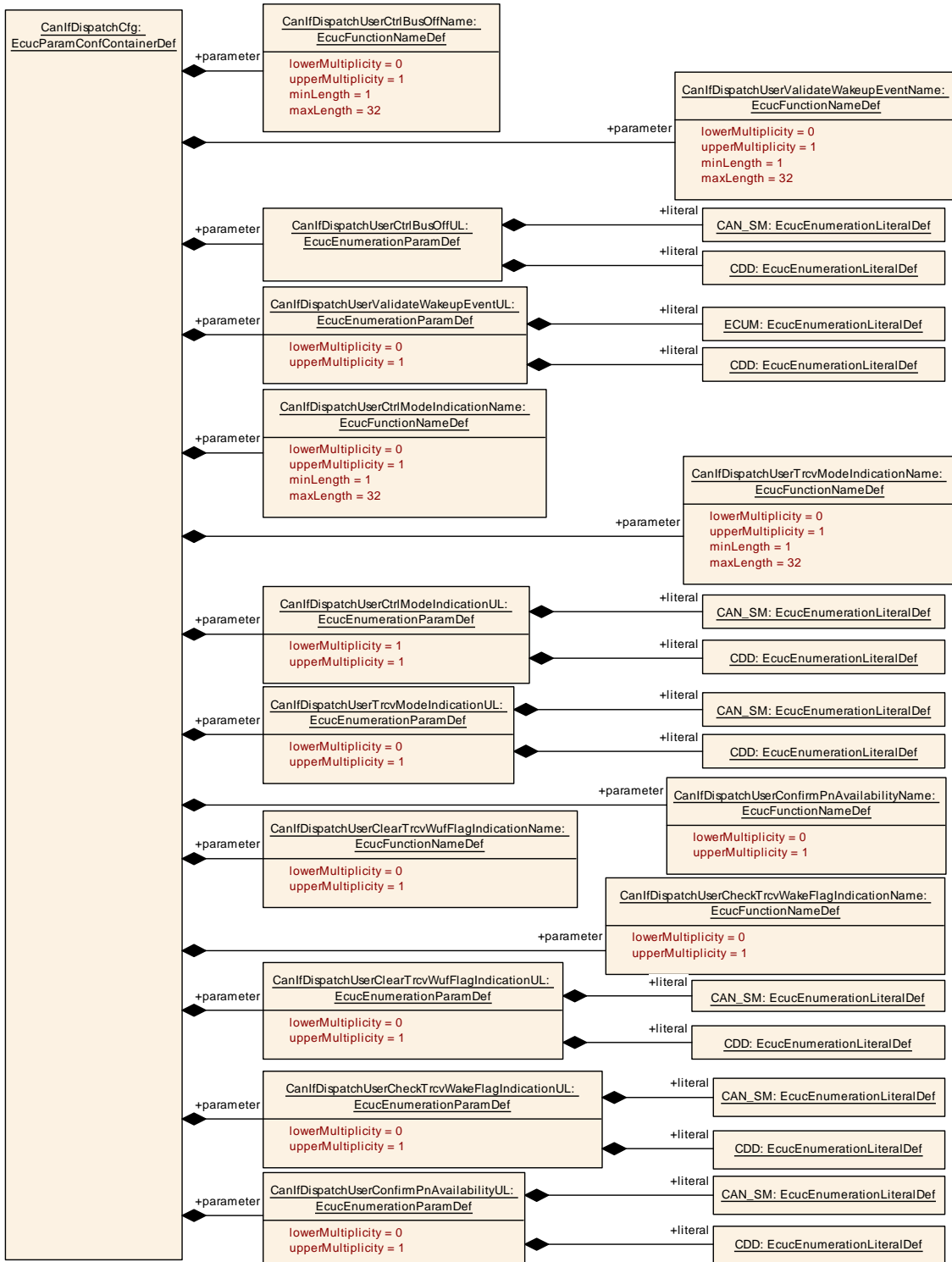


Figure 10.10: AR\_EcucDef\_CanIfDispatchCfg

### 10.1.11 CanIfCtrlCfg

<b>SWS Item</b>	[ECUC_CanIf_00546]		
<b>Container Name</b>	CanIfCtrlCfg		
<b>Parent Container</b>	<a href="#">CanIfCtrlDrvCfg</a>		
<b>Description</b>	This container contains the configuration (parameters) of an addressed CAN controller by an underlying CAN Driver module. This container is configurable per CAN controller.		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Configuration Parameters</b>			

<b>SWS Item</b>	[ECUC_CanIf_00647]		
<b>Parameter Name</b>	CanIfCtrlId		
<b>Parent Container</b>	<a href="#">CanIfCtrlCfg</a>		
<b>Description</b>	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		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
<b>Range</b>	0 .. 255		
<b>Default value</b>	–		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	[ECUC_CanIf_00637]		
<b>Parameter Name</b>	CanIfCtrlWakeupSupport		
<b>Parent Container</b>	<a href="#">CanIfCtrlCfg</a>		
<b>Description</b>	This parameter defines if a respective controller of the referenced CAN Driver modules is queriable for wake up events. True: Enabled False: Disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	[ECUC_CanIf_00636]		
<b>Parameter Name</b>	CanIfCtrlCanCtrlRef		
<b>Parent Container</b>	CanIfCtrlCfg		
<b>Description</b>	<p>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: CanControllerId, CanWakeupSourceRef</p> <p>Range: 0..max. number of underlying supported CAN controllers</p>		
<b>Multiplicity</b>	1		
<b>Type</b>	Symbolic name reference to CanController		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	-	
<b>Scope / Dependency</b>	scope: ECU dependency: amount of CAN controllers		

**No Included Containers**

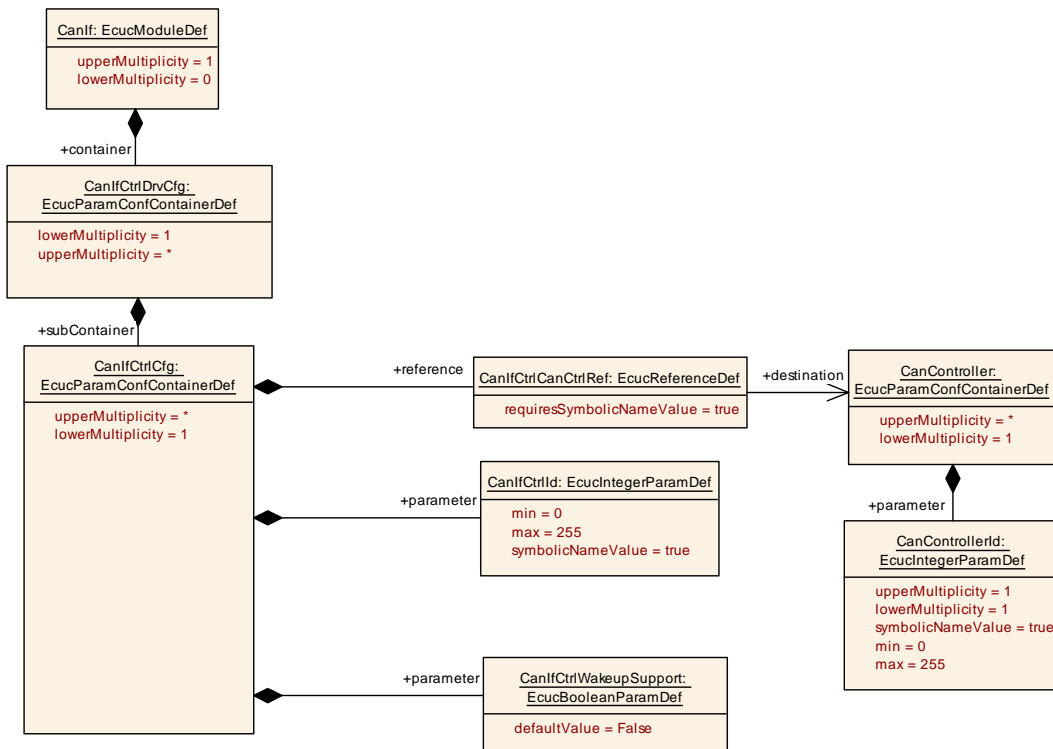


Figure 10.11: AR\_EcucDef\_CanIfCtrlCfg

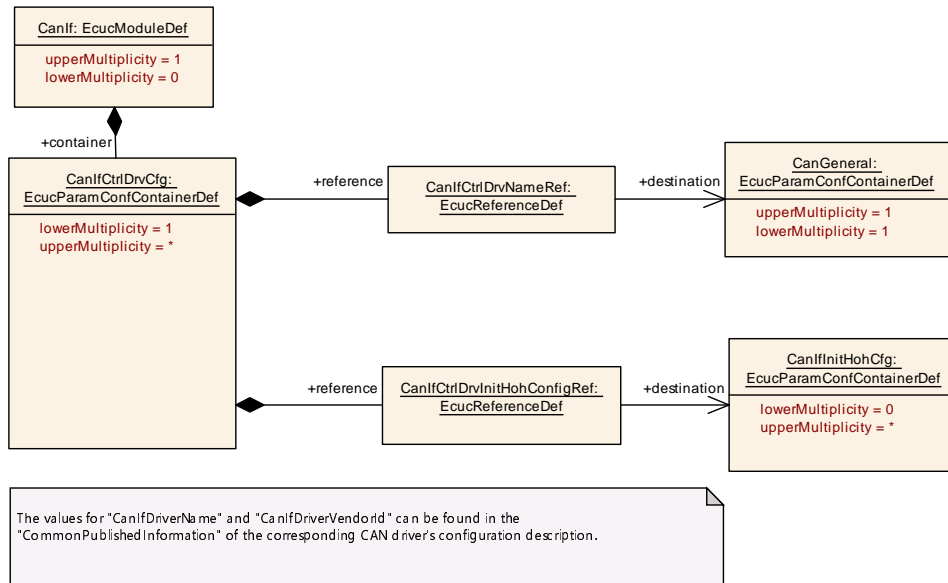
### 10.1.12 CanIfCtrlDrvCfg

<b>SWS Item</b>	[ECUC_CanIf_00253]		
<b>Container Name</b>	CanIfCtrlDrvCfg		
<b>Parent Container</b>	<a href="#">CanIf</a>		
<b>Description</b>	Configuration parameters for all the underlying CAN Driver modules are aggregated under this container. For each CAN Driver module a separate instance of this container has to be provided.		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Configuration Parameters</b>			

<b>SWS Item</b>	[ECUC_CanIf_00642]		
<b>Parameter Name</b>	CanIfCtrlDrvInitHohConfigRef		
<b>Parent Container</b>	<a href="#">CanIfCtrlDrvCfg</a>		
<b>Description</b>	Reference to the Init Hoh Configuration		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to <a href="#">CanIfInitHohCfg</a>		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	[ECUC_CanIf_00638]		
<b>Parameter Name</b>	CanIfCtrlDrvNameRef		
<b>Parent Container</b>	<a href="#">CanIfCtrlDrvCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to CanGeneral		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
<a href="#">CanIfCtrlCfg</a>	1..*	This container contains the configuration (parameters) of an addressed CAN controller by an underlying CAN Driver module. This container is configurable per CAN controller.


**Figure 10.12: AR\_EcucDef\_CanIfCtrlDrvCfg**

### 10.1.13 CanIfTrcvDrvCfg

<b>SWS Item</b>	[ECUC_CanIf_00273]		
<b>Container Name</b>	CanIfTrcvDrvCfg		
<b>Parent Container</b>	CanIf		
<b>Description</b>	This container contains the configuration (parameters) of all addressed CAN transceivers by each underlying CAN Transceiver Driver module. For each CAN transceiver Driver a separate instance of this container shall be provided.		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Configuration Parameters</b>			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
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 separate instance of this container has to be provided.

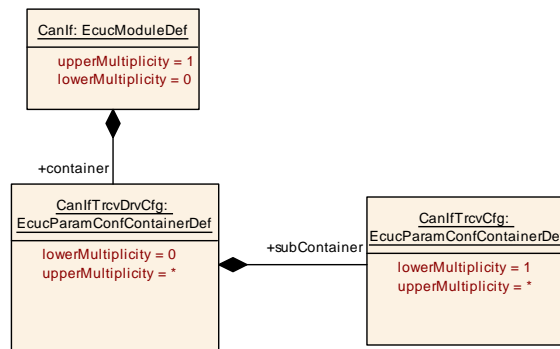


Figure 10.13: AR\_EcucDef\_CanIfTrcvDrvCfg

### 10.1.14 CanIfTrcvCfg

<b>SWS Item</b>	[ECUC_CanIf_00587]		
<b>Container Name</b>	CanIfTrcvCfg		
<b>Parent Container</b>	<a href="#">CanIfTrcvDrvCfg</a>		
<b>Description</b>	This container contains the configuration (parameters) of one addressed CAN transceiver by the underlying CAN Transceiver Driver module. For each CAN transceiver a separate instance of this container has to be provided.		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Configuration Parameters</b>			

<b>SWS Item</b>	[ECUC_CanIf_00654]		
<b>Parameter Name</b>	CanIfTrcvId		
<b>Parent Container</b>	<a href="#">CanIfTrcvCfg</a>		
<b>Description</b>	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		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
<b>Range</b>	0 .. 255		
<b>Default value</b>	–		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		



<b>SWS Item</b>	<b>[ECUC_CanIf_00606]</b>		
<b>Parameter Name</b>	CanIfTrcvWakeupSupport		
<b>Parent Container</b>	<a href="#">CanIfTrcvCfg</a>		
<b>Description</b>	<p>This parameter defines if a respective transceiver of the referenced CAN Transceiver Driver modules is queriable for wake up events.</p> <p>True: Enabled False: Disabled</p>		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00605]</b>		
<b>Parameter Name</b>	CanIfTrcvCanTrcvRef		
<b>Parent Container</b>	<a href="#">CanIfTrcvCfg</a>		
<b>Description</b>	<p>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.</p> <p>Range: 0..max. number of underlying supported CAN transceivers</p>		
<b>Multiplicity</b>	1		
<b>Type</b>	Symbolic name reference to CanTrcvChannel		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: ECU dependency: amount of CAN transceivers		

<b>No Included Containers</b>
-------------------------------

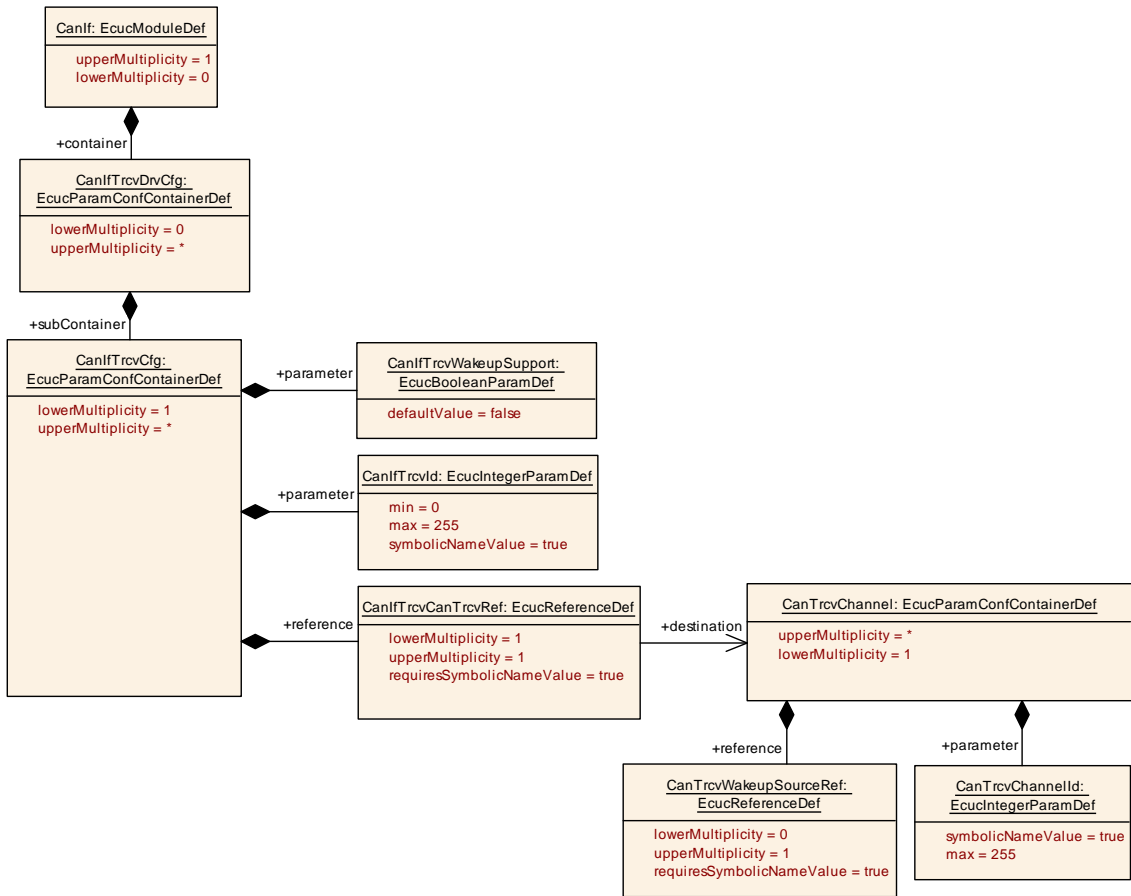


Figure 10.14: AR\_EcucDef\_CanIfTrcvCfg

### 10.1.15 CanIfInitHohCfg

<b>SWS Item</b>	[ECUC_CanIf_00257]		
<b>Container Name</b>	CanIfInitHohCfg		
<b>Parent Container</b>	CanIfInitCfg		
<b>Description</b>	This container contains the references to the configuration setup of each underlying CAN Driver.		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Configuration Parameters</b>			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
CanIfHrhCfg	0..*	This container contains configuration parameters for each hardware receive object (HRH).
CanIfHthCfg	0..*	This container contains parameters related to each HTH.

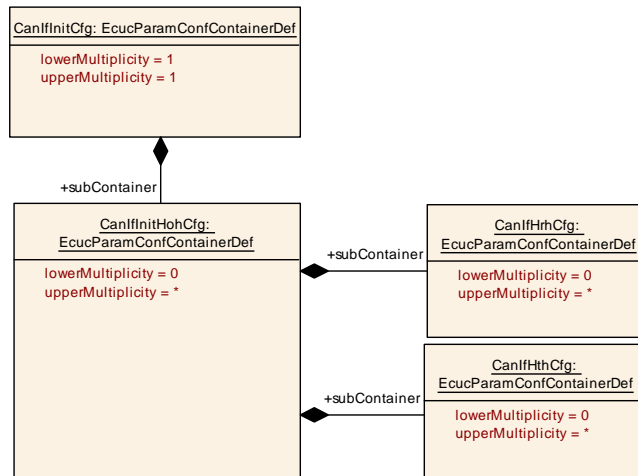


Figure 10.15: AR\_EcucDef\_CanflnitHohCf

### 10.1.16 CanflHthCf

SWS Item	[ECUC_Canlf_00258]		
Container Name	CanflHthCf		
Parent Container	CanflnitHohCf		
Description	This container contains parameters related to each HTH.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	[ECUC_Canlf_00625]		
Parameter Name	CanflHthCanCtrlIdRef		
Parent Container	CanflHthCf		
Description	Reference to controller Id to which the HTH belongs to. A controller can contain one or more HTHs.		
Multiplicity	1		
Type	Reference to CanflCtrlCf		
Post-Build Variant Value	true		
Value Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

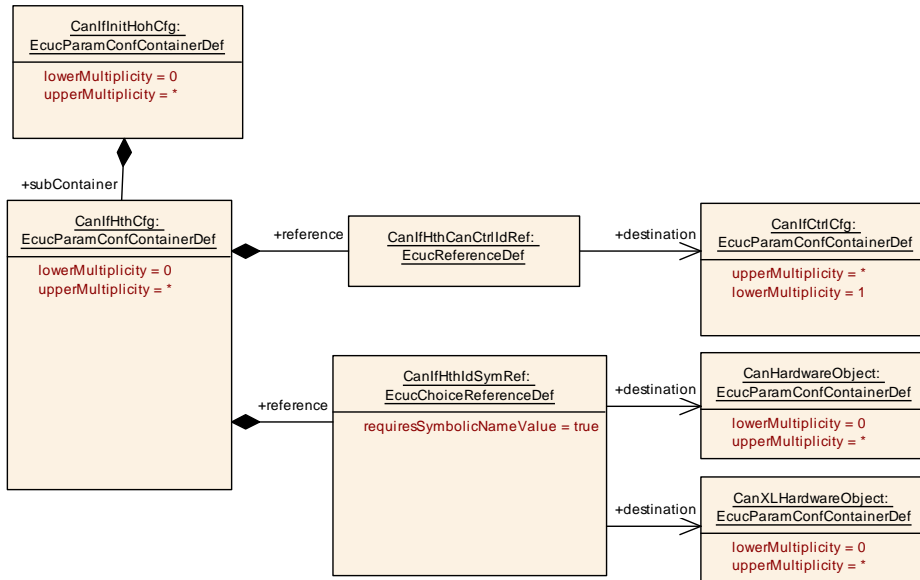
SWS Item	[ECUC_Canlf_00627]		
Parameter Name	CanflHthIdSymRef		
Parent Container	CanflHthCf		





<b>Description</b>	The parameter refers to a particular HTH object in the CanDrv configuration (see CanHardwareObject or CanXLHardwareObject). CanIf receives the following information of the CanDrv module by this reference: <ul style="list-style-type: none"> <li>• CanHandleType (only for CAN 2.0 and CAN FD)</li> <li>• CanObjectId</li> </ul>		
<b>Multiplicity</b>	1		
<b>Type</b>	Choice symbolic name reference to [ CanHardwareObject, CanXLHardwareObject ]		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

**No Included Containers**



**Figure 10.16: AR\_EcucDef\_CanIfHthCfg**

**10.1.17 CanIfHrhCfg**

<b>SWS Item</b>	[ECUC_CanIf_00259]		
<b>Container Name</b>	CanIfHrhCfg		
<b>Parent Container</b>	CanIfInitHohCfg		
<b>Description</b>	This container contains configuration parameters for each hardware receive object (HRH).		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME





	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Configuration Parameters</b>			

<b>SWS Item</b>	<b>[ECUC_CanIf_00632]</b>		
<b>Parameter Name</b>	CanIfHrhSoftwareFilter		
<b>Parent Container</b>	<a href="#">CanIfHrhCfg</a>		
<b>Description</b>	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 disabled		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucBooleanParamDef		
<b>Default value</b>	true		
<b>Post-Build Variant Value</b>	false		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00631]</b>		
<b>Parameter Name</b>	CanIfHrhCanCtrlIdRef		
<b>Parent Container</b>	<a href="#">CanIfHrhCfg</a>		
<b>Description</b>	Reference to controller Id to which the HRH belongs to. A controller can contain one or more HRHs.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to <a href="#">CanIfCtrlCfg</a>		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

<b>SWS Item</b>	<b>[ECUC_CanIf_00634]</b>		
<b>Parameter Name</b>	CanIfHrhIdSymRef		
<b>Parent Container</b>	<a href="#">CanIfHrhCfg</a>		
<b>Description</b>	The parameter refers to a particular HRH object in the CanDrv configuration (see CanHardwareObject or CanXLHardwareObject).  CanIf receives the following information of the CanDrv module by this reference: <ul style="list-style-type: none"> <li>• CanHandleType (only CAN 2.0 and CAN FD)</li> <li>• CanObjectId</li> </ul>		
<b>Multiplicity</b>	1		
<b>Type</b>	Choice symbolic name reference to [ CanHardwareObject, CanXLHardwareObject ]		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: ECU		

Included Containers		
Container Name	Multiplicity	Scope / Dependency
CanIfHrhRangeCfg	0..*	Defines the parameters required for configuring multiple CANID ranges for a given same HRH.

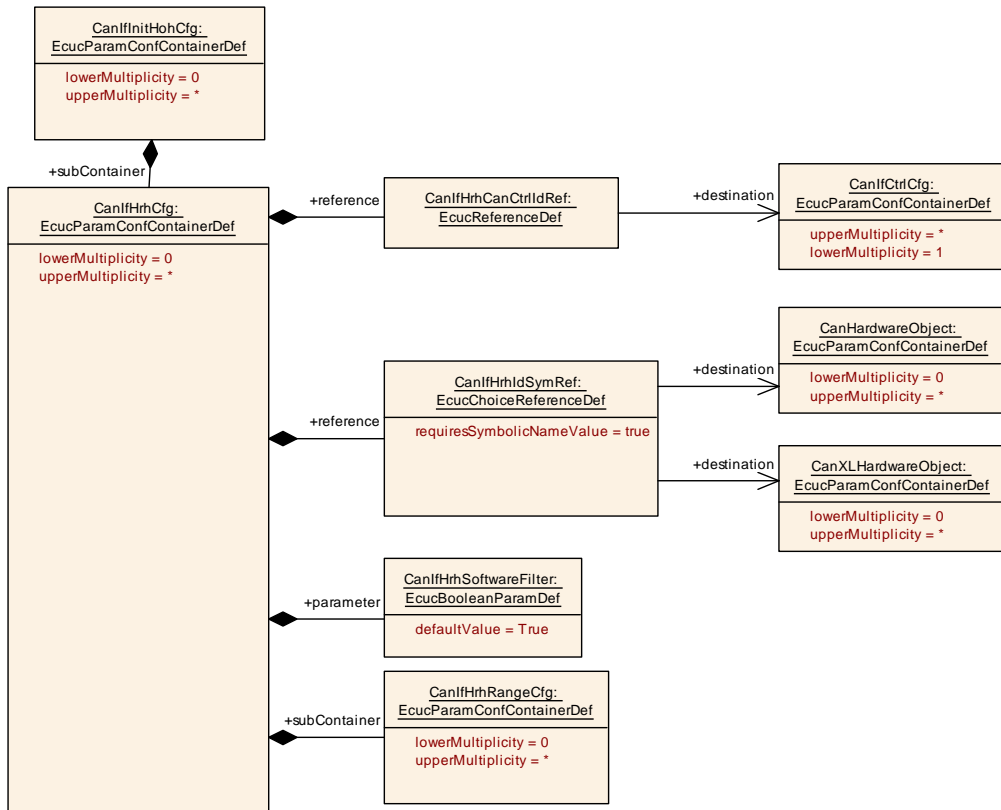


Figure 10.17: AR\_EcucDef\_CanIfHrhCfg

### 10.1.18 CanIfHrhRangeCfg

SWS Item	[ECUC_CanIf_00628]		
Container Name	CanIfHrhRangeCfg		
Parent Container	CanIfHrhCfg		
Description	Defines the parameters required for configuring multiple CANID ranges for a given same HRH.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

<b>SWS Item</b>	<b>[ECUC_CanIf_00825]</b>		
<b>Parameter Name</b>	CanIfHrhRangeBaselId		
<b>Parent Container</b>	<a href="#">CanIfHrhRangeCfg</a>		
<b>Description</b>	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.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	–		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00826]</b>		
<b>Parameter Name</b>	CanIfHrhRangeMask		
<b>Parent Container</b>	<a href="#">CanIfHrhRangeCfg</a>		
<b>Description</b>	Used as mask value in combination with CanIfHrhRangeBaselId for a masked ID range in which all CAN Ids shall pass the software filtering. The size of this parameter is limited by CanIfHrhRangeRxPduRangeCanIdType.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	–		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00629]</b>		
<b>Parameter Name</b>	CanIfHrhRangeRxPduLowerCanId		
<b>Parent Container</b>	<a href="#">CanIfHrhRangeCfg</a>		
<b>Description</b>	Lower CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids shall pass the software filtering.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		





<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00644]</b>		
<b>Parameter Name</b>	CanIfHrhRangeRxPduRangeCanIdType		
<b>Parent Container</b>	<a href="#">CanIfHrhRangeCfg</a>		
<b>Description</b>	Specifies whether a configured Range of CAN Ids shall only consider standard CAN Ids or extended CAN Ids.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucEnumerationParamDef		
<b>Range</b>	EXTENDED	All the CANIDs are of type extended only (29 bit).	
	STANDARD	All the CANIDs are of type standard only (11bit).	
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00630]</b>		
<b>Parameter Name</b>	CanIfHrhRangeRxPduUpperCanId		
<b>Parent Container</b>	<a href="#">CanIfHrhRangeCfg</a>		
<b>Description</b>	Upper CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids shall pass the software filtering.		
<b>Multiplicity</b>	0..1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 536870911		
<b>Default value</b>	-		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Post-Build Variant Value</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

<b>No Included Containers</b>
-------------------------------



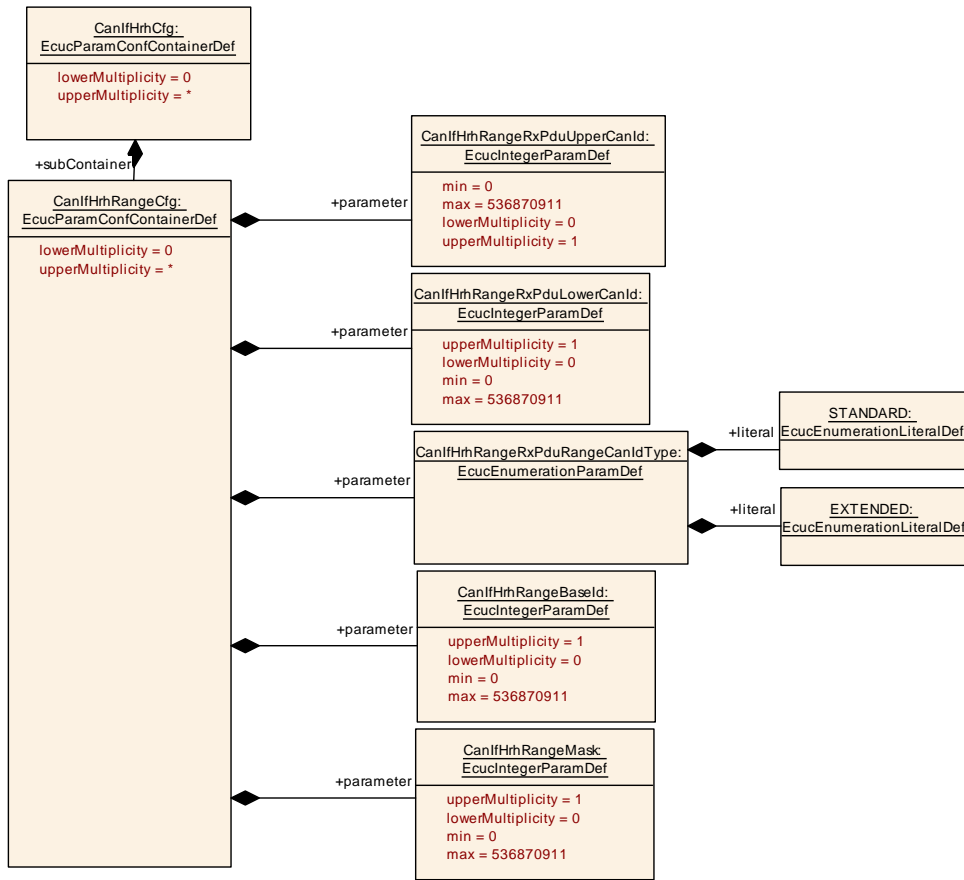


Figure 10.18: AR\_EcucDef\_CanIfHrhRangeCfg

### 10.1.19 CanIfBufferCfg

<b>SWS Item</b>	[ECUC_CanIf_00832]		
<b>Container Name</b>	CanIfBufferCfg		
<b>Parent Container</b>	<a href="#">CanIfInitCfg</a>		
<b>Description</b>	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.		
<b>Post-Build Variant Multiplicity</b>	true		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Configuration Parameters</b>			
<b>SWS Item</b>	[ECUC_CanIf_00834]		
<b>Parameter Name</b>	CanIfBufferSize		
<b>Parent Container</b>	<a href="#">CanIfBufferCfg</a>		

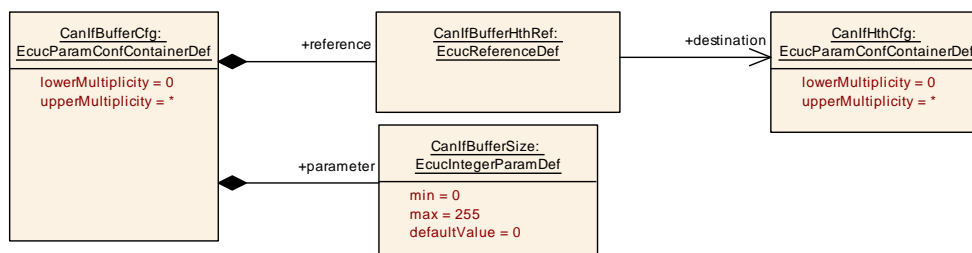




<b>Description</b>	This parameter defines the number of CanIf Tx L-PDUs which can be buffered in one Txbuffer. If this value equals 0, the CanIf does not perform Txbuffering for the CanIf Tx L-PDUs which are assigned to this Txbuffer. If CanIfPublicTxBuffering equals False, this parameter equals 0 for all TxBuffer. If the CanHandleType of the referred HTH equals FULL, this parameter equals 0 for this TxBuffer.		
<b>Multiplicity</b>	1		
<b>Type</b>	EcucIntegerParamDef		
<b>Range</b>	0 .. 255		
<b>Default value</b>	0		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local dependency: CanIfPublicTxBuffering, CanHandleType		

<b>SWS Item</b>	<b>[ECUC_CanIf_00833]</b>		
<b>Parameter Name</b>	CanIfBufferHthRef		
<b>Parent Container</b>	CanIfBufferCfg		
<b>Description</b>	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.		
<b>Multiplicity</b>	1		
<b>Type</b>	Reference to <a href="#">CanIfHthCfg</a>		
<b>Post-Build Variant Value</b>	true		
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	VARIANT-PRE-COMPILE
	<b>Link time</b>	X	VARIANT-LINK-TIME
	<b>Post-build time</b>	X	VARIANT-POST-BUILD
<b>Scope / Dependency</b>	scope: local		

**No Included Containers**



**Figure 10.19: AR\_EcucDef\_CanIfBufferCfg**

### 10.1.20 CanIfSecurityEventRefs

<b>SWS Item</b>	<b>[ECUC_CanIf_00849]</b>		
<b>Container Name</b>	CanIfSecurityEventRefs		
<b>Parent Container</b>	<a href="#">CanIfPublicCfg</a>		
<b>Description</b>	Container for the references to IdsMEvent elements representing the security events that the CanIf module shall report to the IdsM in case the corresponding 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. <b>Tags:</b> atp.Status=draft		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Configuration Parameters</b>			

<b>SWS Item</b>	<b>[ECUC_CanIf_00853]</b>		
<b>Parameter Name</b>	CANIF_SEV_ERRORSTATE_BUSOFF		
<b>Parent Container</b>	<a href="#">CanIfSecurityEventRefs</a>		
<b>Description</b>	The CAN controller transitioned to state busoff. <b>Tags:</b> atp.Status=draft		
<b>Multiplicity</b>	0..1		
<b>Type</b>	Symbolic name reference to IdsMEvent		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00852]</b>		
<b>Parameter Name</b>	CANIF_SEV_ERRORSTATE_PASSIVE		
<b>Parent Container</b>	<a href="#">CanIfSecurityEventRefs</a>		
<b>Description</b>	A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity. <b>Tags:</b> atp.Status=draft		
<b>Multiplicity</b>	0..1		
<b>Type</b>	Symbolic name reference to IdsMEvent		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants





	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00851]</b>		
<b>Parameter Name</b>	CANIF_SEV_RX_ERROR_DETECTED		
<b>Parent Container</b>	<a href="#">CanIfSecurityEventRefs</a>		
<b>Description</b>	A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity. <b>Tags:</b> atp.Status=draft		
<b>Multiplicity</b>	0..1		
<b>Type</b>	Symbolic name reference to IdsMEvent		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>SWS Item</b>	<b>[ECUC_CanIf_00850]</b>		
<b>Parameter Name</b>	CANIF_SEV_TX_ERROR_DETECTED		
<b>Parent Container</b>	<a href="#">CanIfSecurityEventRefs</a>		
<b>Description</b>	A transmission related error was detected. Depending on the context data this could indicate suspicious CAN activity. <b>Tags:</b> atp.Status=draft		
<b>Multiplicity</b>	0..1		
<b>Type</b>	Symbolic name reference to IdsMEvent		
<b>Post-Build Variant Multiplicity</b>	false		
<b>Post-Build Variant Value</b>	false		
<b>Multiplicity Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Value Configuration Class</b>	<b>Pre-compile time</b>	X	All Variants
	<b>Link time</b>	–	
	<b>Post-build time</b>	–	
<b>Scope / Dependency</b>	scope: local		

<b>No Included Containers</b>
-------------------------------

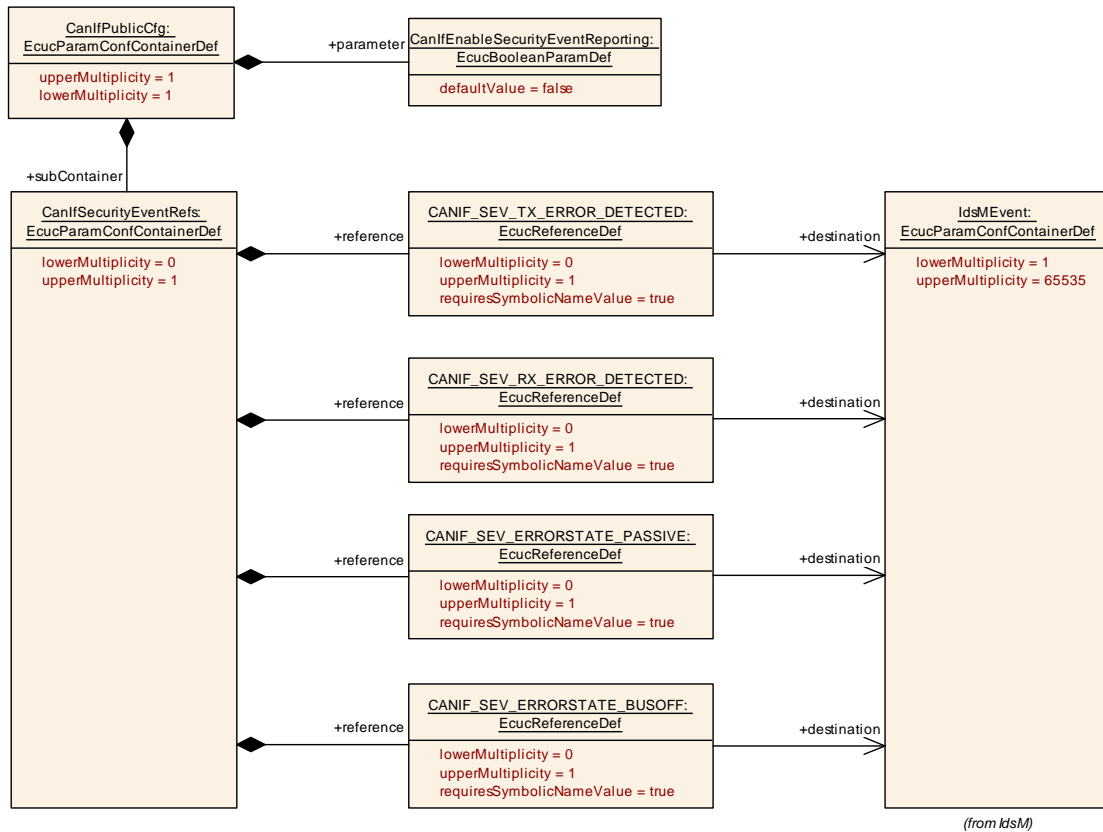


Figure 10.20: AR\_EcucDef\_CanIfSecurityEventRefs

## 10.2 Configuration constraints

**[SWS\_CANIF\_CONSTR\_00001]** [For each [CanIfRxPduCfg](#) that contains [CanIfRxPduXLParams](#), 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](#).] ([SRS\\_Can\\_02003](#))

**[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](#).] ([SRS\\_Can\\_02003](#))

**[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](#).] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[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`.] ([SRS\\_Can\\_02003](#))

**[SWS\_CANIF\_CONSTR\_00011]** [For each `CanIfRxPduCfg` that contains `CanIfRxPduXLParams` where the `CanIfRxPduXLParams.CanIfXLSduType` is configured but not set to `03h`, the parameters `CanIfRxPduCanId`, `CanIfRxPduCanIdType`, 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`.] ([SRS\\_Can\\_02003](#))

**[SWS\_CANIF\_CONSTR\_00012]** [For each `CanIfTxPduCfg` that contains `CanIfTxPduXLParams` where the `CanIfTxPduXLParams.CanIfXLSduType` is configured but not set to 03h, the parameters `CanIfTxPduCanId`, `CanIfTxPduCanIdType`, 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`.] ([SRS\\_Can\\_02003](#))

**[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.] ([SRS\\_Can\\_02003](#))

**[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.] ([SRS\\_Can\\_02003](#))

## A Not applicable requirements

**[SWS\_CANIF\_NA\_00999]** [These requirements are not applicable to this specification.] (*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\_00336, SRS\_BSW\_00417, SRS\_BSW\_00164, SRS\_BSW\_00007, SRS\_BSW\_00307, SRS\_BSW\_00373, SRS\_BSW\_00328, SRS\_BSW\_00378, SRS\_BSW\_00306, SRS\_BSW\_00308, SRS\_BSW\_00309, SRS\_BSW\_00330, SRS\_BSW\_00172, SRS\_BSW\_00010, SRS\_BSW\_00341, SRS\_BSW\_00334, SRS\_Can\_01139, SRS\_Can\_01014*)