|  |  |
| --- | --- |
| **Document Title** | Specification of CAN Interface |
| **Document Owner** | AUTOSAR |
| **Document Responsibility** | AUTOSAR |
| **Document Identification No** | 12 |

|  |  |
| --- | --- |
| **Document Status** | published |
| **Part of AUTOSAR Standard** | Classic Platform |
| **Part of Standard Release** | R21-11 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Change History** | | | |
| **Date** | **Release** | **Changed by** | **Description** |
| 2021-11-25 | R21-11 | AUTOSAR  Release Management | * Support hardware-based timestamping * Minor corrections/clarifications * Editorial changes |
| 2020-11-30 | R20-11 | AUTOSAR  Release Management | * Introduction of Security Events * Rework of Error classification chapters * Removal of pretended networking * Minor corrections/clarifications |
| 2019-11-28 | R19-11 | AUTOSAR  Release Management | * Update reference to ISO 11898-1:2015 * Minor corrections * Editorial changes * Changed Document Status from Final to published |
| 2018-10-31 | 4.4.0 | AUTOSAR  Release Management | * BusMirroring (CONC\_634) * Receive Data Length Check per Pdu * Remove dummy implementations for Cancel Transmit APIs * Header File Cleanup |
| 2017-12-08 | 4.3.1 | AUTOSAR  Release Management | * Introduction of Runtime errors * Replace Can\_ReturnType with Std\_ReturnType overlay * Minor corrections * Editorial changes |
| 2016-11-30 | 4.3.0 | AUTOSAR  Release Management | * Remove CCMSM * Rework MetaData handling * Reliable TxConfirmation * Error Active/Passive State API |
| 2015-07-31 | 4.2.2 | AUTOSAR  Release Management | * Clarified wakeup, buffering, transmit, and variants * Removed deprecated APIs * Editorial changes |
| 2014-10-31 | 4.2.1 | AUTOSAR  Release Management | * Full CAN FD Support * Global Time Synchronization over CAN * Removed CanIf\_CancelTxConfirmation * Small improvements |
| 2014-03-31 | 4.1.3 | AUTOSAR  Release Management | * Removed BSW Exclusive areas * Set ICOM support to optional * Can\_IdType handling * Small improvements |
| 2013-10-31 | 4.1.2 | AUTOSAR  Release Management | * Restricted PDU mode changes * Removed critical section handling description in [chapter](#_bookmark438) [9](#_bookmark438) * Set CanIfInitRefCfgSet oboslete * Pretended Networking section * Small improvements |
| 2013-03-15 | 4.1.1 | AUTOSAR  Administration | * CAN FD (without DLC extension) * Pretended Networking (ICOM) * Heavy Duty Vehicle (J1939) support * PduModes and PnTxFilter for clean wake-up * Relation between PDUs & HOHs * Post-build loadable concept |
| 2011-12-22 | 4.0.3 | AUTOSAR  Administration | * Partial Networking Support * Improved Transmit Buffering * Improved Error Detection |
| 2009-12-18 | 4.0.1 | AUTOSAR  Administration | * Updated chapters "Version Checking" and "Published Information" * Multiple CAN IDs could optionally be assigned to one I-PDU * Wake-up validation optionally only via NM PDUs * Asynch. mode indication call-backs instead of synch. mode changes * No automatic PDU channel mode change when CC mode changes * TxConfirmation state entered for BusOff Recovery * WakeupSourceRefIn and WakeupSourceRefOut * PduInfoPtr instead of SduDataPtr * Introduction of Can\_GeneralTypes.h and Can\_HwHandleType * Transceiver types of chapter 8. shifted to transceiver SWS |
| 2010-02-02 | 3.1.4 | AUTOSAR  Administration | * HOH definition * abstracted ControllerId and TransceiverId * No changing of baudrate via CanIf and CanIf\_ControllerInit * Dispatcher adapted because of CDD * TxBuffering: only one buffer per L-PDU * Wake up mechanism adapted to environment behavior (network -> controller/transceiver; wakeupSource) * Mode changes made asynchronous * no complete state machine in CanIf, just buffered states per controller * Legal disclaimer revised |
| 2008-08-13 | 3.1.1 | AUTOSAR  Administration | Legal disclaimer revised |
| 2008-02-01 | 3.0.2 | AUTOSAR  Administration | * Replaced chapter 10 content with generated tables from AUTOSAR MetaModel. |
| 2008-02-01 | 3.0.2 | AUTOSAR  Administration | * Interface abstraction: network related interface changed into a controller related one * Wakeup mechanism completely reworked, APIs added & changed for Wakeup * Initialization changed (flat initialization) * Scheduled main functions skipped due to changed BSW Scheduler responsibility * Document meta information extended * Small layout adaptations made |
| 2007-12-21 | 3.0.1 | AUTOSAR  Administration | * Header file structure changed * Support of mixed mode operation (Standard CAN & Extended CAN in parallel on one network) added * Support of CAN Transceiver API   <User>\_DlcErrorNotification deleted   * Pre-compile/Link-Time/Post-Built definition for configuration parameters partly changed * Re-entrant interface call allowed for certain APIs * Support of AUTOSAR BSW Scheduler added * Support of memory mapping added * Configuration container structure reworked * Various of clarification extensions and corrections |
| 2006-05-16 | 2.0.0 | AUTOSAR  Administration | Second Release |
| 2005-05-31 | 1.0.0 | AUTOSAR  Administration | Initial Release |

## Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## Table of Contents

[Disclaimer 5](#_Toc167012323)

[Table of Contents 6](#_Toc167012324)

[1. Introduction and functional overview 9](#_Toc167012325)

[2. Acronyms and Abbreviations 11](#_Toc167012326)

[3. Related documentation 14](#_Toc167012327)

[3.1. Input documents & related standards and norms 14](#_Toc167012328)

[3.2. Related specification 14](#_Toc167012329)

[4. Constraints and assumptions 15](#_Toc167012330)

[4.1. Limitations 15](#_Toc167012331)

[4.2. Applicability to car domains 15](#_Toc167012332)

[5. Dependencies to other modules 16](#_Toc167012333)

[5.1. Upper Protocol Layers 17](#_Toc167012334)

[5.2. Initialization: Ecu State Manager 17](#_Toc167012335)

[5.3. Mode Control: CAN State Manager 17](#_Toc167012336)

[5.4. Lower layers: CAN Driver 17](#_Toc167012337)

[5.5. Lower layers: CAN Transceiver Driver 18](#_Toc167012338)

[5.6. Configuration 18](#_Toc167012339)

[5.7. File structure 20](#_Toc167012340)

[6. Requirements Tracing 21](#_Toc167012341)

[7. Functional specification 25](#_Toc167012342)

[7.1. General Functionality 25](#_Toc167012343)

[7.2. Hardware object handles 25](#_Toc167012344)

[7.3. Static L-PDUs 28](#_Toc167012345)

[7.4. Dynamic L-PDUs 29](#_Toc167012346)

[7.5. Physical channel view 30](#_Toc167012347)

[7.6. CAN Hardware Unit 33](#_Toc167012348)

[7.7. BasicCAN and FullCAN reception 34](#_Toc167012349)

[7.8. Initialization 35](#_Toc167012350)

[7.9. Transmit request 36](#_Toc167012351)

[7.10. Transmit data flow 36](#_Toc167012352)

[7.11. Transmit buffering 39](#_Toc167012353)

[7.12. Transmit confirmation 42](#_Toc167012354)

[7.13. Receive data flow 42](#_Toc167012355)

[7.14. Receive indication 46](#_Toc167012356)

[7.15. Read received data 47](#_Toc167012357)

[7.16. Read Tx/Rx notification status 48](#_Toc167012358)

[7.17. Data integrity 48](#_Toc167012359)

[7.18. CAN Controller Mode 49](#_Toc167012360)

[7.19. PDU channel mode control 54](#_Toc167012361)

[7.20. Software receive filter 58](#_Toc167012362)

[7.21. Data Length Check 60](#_Toc167012363)

[7.22. L-SDU dispatcher to upper layers 60](#_Toc167012364)

[7.23. Polling mode 61](#_Toc167012365)

[7.24. Multiple CAN Driver support 61](#_Toc167012366)

[7.25. Partial Networking 67](#_Toc167012367)

[7.26. CAN FD Support 67](#_Toc167012368)

[7.27. Security Events 69](#_Toc167012369)

[7.28. Error classification 71](#_Toc167012370)

[8. API specification 74](#_Toc167012371)

[8.1. Imported types 74](#_Toc167012372)

[8.2. Type definitions 75](#_Toc167012373)

[8.3. Function definitions 78](#_Toc167012374)

[8.4. Callback notifications 106](#_Toc167012375)

[8.5. Scheduled functions 115](#_Toc167012376)

[8.6. Expected interfaces 115](#_Toc167012377)

[9. Sequence diagrams 125](#_Toc167012378)

[9.1. Transmit request (single CAN Driver) 125](#_Toc167012379)

[9.2. Transmit request (multiple CAN Drivers) 126](#_Toc167012380)

[9.3. Transmit confirmation (interrupt mode) 128](#_Toc167012381)

[9.4. Transmit confirmation (polling mode) 129](#_Toc167012382)

[9.5. Transmit confirmation (with buffering) 131](#_Toc167012383)

[9.6. Trigger Transmit Request 133](#_Toc167012384)

[9.7. Receive indication (interrupt mode) 135](#_Toc167012385)

[9.8. Receive indication (polling mode) 138](#_Toc167012386)

[9.9. Read received data 140](#_Toc167012387)

[9.10. Start CAN network 144](#_Toc167012388)

[9.11. BusOff notification 146](#_Toc167012389)

[9.12. BusOff recovery 147](#_Toc167012390)

[10. Configuration specification 149](#_Toc167012391)

[10.1. Containers and configuration parameters 149](#_Toc167012392)

[A - Not applicable requirements 1](#_Toc167012393)

### Introduction and functional overview

Tài liệu này mô tả chức năng, API và cấu hình cho module AUTOSAR Basic Software CAN Interface.

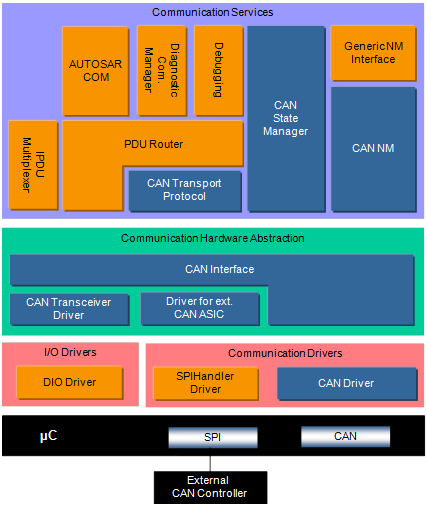
Như minh họa trong [Figure](#_bookmark1) [1.1](#_bookmark1), mô-đun CAN Interface nằm giữa các thiết bị CAN drivers (CAN Driver [[1](#_bookmark36)] and Transceiver Driver [[2](#_bookmark37)]) và các lớp dịch vụ giao tiếp trên như (i.e. CAN State Manager [[3](#_bookmark38)], CAN Network Management [[4](#_bookmark39)], CAN Transport Protocol [[5](#_bookmark40)], PDU Router [[6](#_bookmark41)]).. Nó đại diện cho giao diện với các dịch vụ của CAN Driver cho các lớp giao tiếp trên cùng.

Mô-đun CAN Interface cung cấp một giao diện duy nhất để quản lý các loại thiết bị phần cứng CAN khác nhau như CAN Controllers và CAN Transceivers được sử dụng bởi bố trí ECU hardware được xác định. Do đó, nhiều CAN Controller- s/CAN Transceivers nội bộ và bên ngoài có thể được điều khiển bởi mô-đun CAN State Managers dựa trên quan điểm liên quan đến kênh CAN vật lý.

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

As depicted in the CAN Interface module is located between the low level CAN device drivers (CAN Driver [[1](#_bookmark36)] and Transceiver Driver [[2](#_bookmark37)]) and the upper commu- nication service layers (i.e. CAN State Manager [[3](#_bookmark38)], CAN Network Management [[4](#_bookmark39)], CAN Transport Protocol [[5](#_bookmark40)], PDU Router [[6](#_bookmark41)]). 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 hard- ware 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**](#_bookmark42)**])**

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](#_bookmark8) 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](#_bookmark24) and thus *waking up* / *participating on a network*. Its data processing and notification API is based on [CAN L-SDUs](#_bookmark4), whereas APIs for control and mode handling provides a [CAN Controller](#_bookmark24) related view.

In case of [Transmit Requests](#_bookmark33) [CanIf](#_bookmark8) completes the [L-PDU](#_bookmark3) transmission with cor- responding parameters and relays the [CAN L-PDU](#_bookmark3) via the appropriate [CanDrv](#_bookmark5) to the

[CAN Controller](#_bookmark24). At reception [CanIf](#_bookmark8) distributes the [Received L-PDUs](#_bookmark26) as [L-](#_bookmark4)

[-SDUs](#_bookmark4) to the upper layer. The assignment between Receive [L-SDU](#_bookmark4) and upper layer is statically configured. At transmit confirmation [CanIf](#_bookmark8) 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.

### Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the CAN Interface module that are not included in the [[8](#_bookmark43), 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 |
| PduR | PDU Router module |
| PN | Partial Networking |
| SchM | Scheduler Module |
| Buffer | Fixed sized memory area for a single data unit (e.g. CAN ID, Data  Length, SDU, etc.) is stored at a dedicated memory address in RAM. |
| CAN communication matrix | Describes the complete CAN network:   * Participating nodes * Definition of all CAN PDUs (Identifier, Data Length)   Source and Sinks for PDUs |
| CAN Controller | A CAN Controller is a CPU on-chip or external standalone hard-  ware 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 Con-  trollers of the same type and one, two or multiple CAN RAM areas. The CAN Hardware Unit is located on-chip or as exter- nal device. The CAN hardware unit is represented by one CAN Driver. |
| CanIf Controller mode state ma- chine | 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 CanI- fTxBuffer 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 hard- ware object. The HRH is used as a parameter by the CAN Inter- face 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 multi- ple 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 pres-  ence of a pending low-priority L-PDU in the same transmit hard- ware object. |
| Integration Code | Code that the Integrator needs to add to an AUTOSAR System,  to adapt non-standardized functionalities. Examples are Call- outs of the ECU State Manager and Callbacks of various other BSW modules. The I/O Hardware Abstraction is called Integra- tion 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 net-  work. 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 be- cause 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 |

### Related documentation

#### Input documents & related standards and norms

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. Specification of ECU Configuration AUTOSAR\_TPS\_ECUConfiguration

#### Related specification

AUTOSAR provides a General Specification on Basic Software modules [[9](#_bookmark44), SWS BSW General], which is also valid for CAN Interface.

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

### Constraints and assumptions

#### Limitations

The CAN Interface can be used for CAN communication only and is specifically de- signed to operate with one or multiple underlying CAN Drivers and CAN Transceiver Drivers. Several CAN Driver modules covering different CAN Hardware Units are rep- resented by just one generic interface as specified in the CAN Driver specification [[1](#_bookmark36)]. 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](#_bookmark37), Specification of CAN Transceiver Driver]. Other protocols than CAN (i.e. LIN or FlexRay) are not supported.

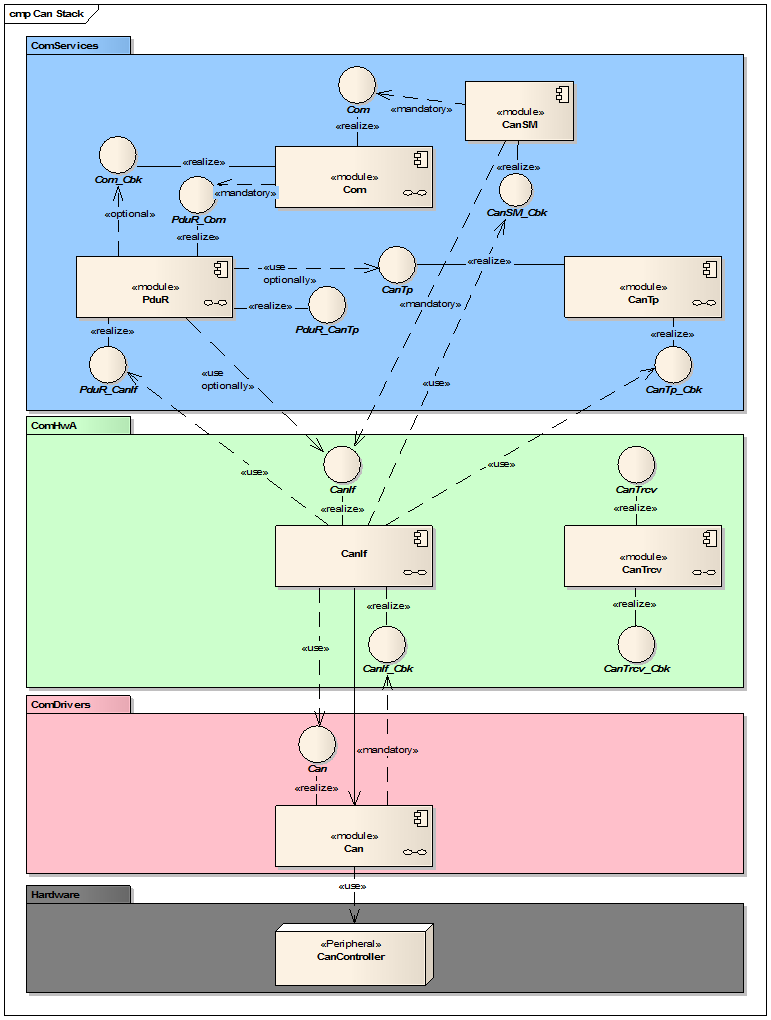
Please be aware that an active PnTxFilter ensures that the first messages on bus is CanIfTxPduPnFilterPdu. In case that CanIfTxPduPnFilterPdu is the NM-PDU the COM-Stack start up takes care that the PduGroups are disabled until successful trans- mission of that PDU. However, transmit requests for other 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 CanIfTx- PduPnFilterPdu. In case communication is ongoing and there is an successful recep- tion 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.

#### Applicability to car domains

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

### Dependencies to other modules

This section describes the relations to other modules within the AUTOSAR basic soft- ware architecture. It contains brief descriptions of configuration information and ser- vices, which are required by the CAN Interface Layer from other modules (see [Figure](#_bookmark55) [5.1](#_bookmark55)).



**Figure 5.1:** **CANIF dependencies in AUTOSAR BSW**

#### Upper Protocol Layers

Inside the AUTOSAR BSW architecture the upper layers of the CAN Interface mod- ule (Abbr.: [CanIf](#_bookmark8)) are represented by the PDU Router module (Abbr.: [PduR](#_bookmark22)), CAN Network Management module (Abbr.: [CanNm](#_bookmark9)), CAN Transport Layer module (Abbr.: [CanTp](#_bookmark11)), CAN State Manager module (Abbr.: [CanSm](#_bookmark10)), ECU State Manager module (Abbr.: [EcuM](#_bookmark16)), Complex Driver modules (Abbr.: *CDD*), Universal Calibration Protocol module (Abbr.: *XCP*), Global Time Synchronization over CAN (Abbr.: [CanTSyn](#_bookmark13)), J1939 Transport Layer module (Abbr.: [J1939Tp](#_bookmark21)) and J1939 Network Management module (Abbr.: [J1939Nm](#_bookmark20)).

The AUTOSAR BSW architecture indicates that the application data buffers are lo- cated in the upper layer, to which they belong. Direct access to these buffers is pro- hibited. The buffer location is passed by the CanIf from or to the CAN Driver module (Abbr.: [CanDrv](#_bookmark5)) during transmission and reception. During execution of these trans- mission/reception indication services buffer location is passed. Data integrity is guar- anteed by use of lock mechanisms each time the buffer has been accessed. See [section](#_bookmark192) [7.17](#_bookmark192) “[Data integrity](#_bookmark192)”.

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.

#### Initialization: Ecu State Manager

The EcuM initializes the CanIf (refer to [[3](#_bookmark38), Specification of ECU State Manager]).

#### Mode Control: CAN State Manager

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

#### Lower layers: CAN Driver

The main lower layer CAN device driver is represented by the CanDrv (see [[1](#_bookmark36), Specifi- cation 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 under- lying CAN Controllers.

[CanDrv](#_bookmark5) provides a normalized L-PDU to ensure hardware independence of [CanIf](#_bookmark8). 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](#_bookmark5). For [CanIf](#_bookmark8) the kind of L-PDU buffer is invisible.

The CanIf provides notification services used by the CanDrv in all notifications sce- narios, for example: *transmit confirmation* ([subsection](#_bookmark375) [8.4.2](#_bookmark375) “[CanIf\_TxConfirmation](#_bookmark375)”, see [[SWS\_CANIF\_00007](#_bookmark376)]), *receive indication* ([subsection](#_bookmark381) [8.4.3](#_bookmark381) “[CanIf\_RxIndication](#_bookmark381)”, see [[SWS\_CANIF\_00006](#_bookmark382)]) and *notification of a controller mode change* ([subsection](#_bookmark400) [8.4.8](#_bookmark400), see [[SWS\_CANIF\_00699](#_bookmark401)]).

In case of using multiple CanDrv serving different interrupt vectors these callback ser- vices mentioned above must be re-entrant, refer to [section](#_bookmark225) [7.24](#_bookmark225) “[Multiple CAN Driver](#_bookmark225) [support](#_bookmark225)”. Reentrancy of callback functions is specified in [section](#_bookmark372) [8.4](#_bookmark372).

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](#_bookmark62) [5.7](#_bookmark62) “[File structure](#_bookmark62)”.

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](#_bookmark12) is sufficient, whereas dependent to the type of the CAN Controller devices one or two different CanDrv are needed (see [section](#_bookmark149) [7.5](#_bookmark149) “[Physical channel view](#_bookmark149)”).

#### Lower layers: CAN Transceiver Driver

The second available lower layer CAN device driver is represented by the CanTrcv (see [[2](#_bookmark37), 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 inter- face. These are further described in the CAN Transceiver Driver SWS (see [[2](#_bookmark37), Specifi- cation 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()](#_bookmark322), see [[SWS\_CANIF\_00287](#_bookmark321)]).

#### Configuration

The [CanIf](#_bookmark8) design is optimized to manage CAN protocol specific capabilities and han- dling of the used underlying CAN Controller.

The CanIf is capable to change the CAN configuration without a *re-build*. Therefore, the function [CanIf\_Init()](#_bookmark257) (see [[SWS\_CANIF\_00001](#_bookmark256)]) retrieves the required CAN con- figuration information from configuration containers and parameters, which are speci- fied (linked as references, or additional parameters) in [chapter](#_bookmark451) [10](#_bookmark451), see [Figure](#_bookmark453) [10.1](#_bookmark453).

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](#_bookmark530)).
* 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](#_bookmark541); [CanIfHthIdSymRef](#_bookmark542)).
* Range of received CAN IDs passing hardware acceptance filter for each hard- ware 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](#_bookmark215) [7.20](#_bookmark215) “[Software receive filter](#_bookmark215)”, see [CanIfHrhSoftware-](#_bookmark545) [Filter](#_bookmark545), [CanIfHrhCanCtrlIdRef](#_bookmark546), [CanIfHrhIdSymRef](#_bookmark547))

[CanIf](#_bookmark8) needs information about all used upper communication service layers and [L-SDUs](#_bookmark4) to be dispatched. The following information has to be set up at configuration time for integration of [CanIf](#_bookmark8) inside the AUTOSAR COM stack:

* Transmitting upper layer module and transmit *I-PDU* for each transmit [L-SDU](#_bookmark4).

=> Used for dispatching of transmit confirmation services (see [CanIfTxPduId](#_bookmark485)).

* Receiving upper layer module and receive *I-PDU* for each receive [L-SDU](#_bookmark4).

=> Used for [L-SDU](#_bookmark4) dispatching during receive indication (see [CanIfRxPduId](#_bookmark501)).

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](#_bookmark481), [CanIfRxPduCfg](#_bookmark495)):

* All L-PDUs received on each physical channel of this ECU.

=> Used for software filtering and receive [L-SDU](#_bookmark4) dispatch

* All [L-SDUs](#_bookmark4) 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](#_bookmark4) (i.e. PduR, CanNm, CanTp).

=> Used for the transmit confirmation services

* Receiver for each receive [L-SDU](#_bookmark4) (i.e. PduR, CanNm, CanTp)

=> Used for the L-PDU dispatch

* Symbolic [L-PDU](#_bookmark3)/[L-SDU](#_bookmark4) name.

=> Used for the representation of Rx/Tx data buffer addresses

#### File structure

##### Code file structure

**[SWS\_CANIF\_00378]** *[*[CanIf](#_bookmark8) shall access the location of the API of all used underly- ing [CanDrvs](#_bookmark5) for link time configuration by a set of function pointers for each [CanDrv](#_bookmark5).*] ()*

The values for the function pointers for each [CanDrv](#_bookmark5) are given at link time.

##### 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](#_bookmark8).*]()*

Constants, global data types and functions that are only used by [CanIf](#_bookmark8) internally, are declared within CanIf.c.

**[SWS\_CANIF\_00903]** *[*[CanIf](#_bookmark8) shall include the header file Mirror.h if Bus Mirroring is enabled (see [CanIfBusMirroringSupport](#_bookmark462)).*]([SRS\_Can\_01172](#_bookmark138))*

### Requirements Tracing

The following tables references the requirements specified in [[10](#_bookmark45)] as well as [[11](#_bookmark46)] 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 | [[SWS\_CANIF\_00913](#_bookmark235)]  [[SWS\_CANIF\_00915](#_bookmark236)] [[SWS\_CANIF\_00916](#_bookmark237)] [[SWS\_CANIF\_00917](#_bookmark238)] [[SWS\_CANIF\_00918](#_bookmark239)] [[SWS\_CANIF\_00919](#_bookmark408)] [[SWS\_CANIF\_00920](#_bookmark412)] [[SWS\_CANIF\_00921](#_bookmark413)] [[SWS\_CANIF\_91008](#_bookmark406)] [[SWS\_CANIF\_91009](#_bookmark410)] [[SWS\_CANIF\_91010](#_bookmark234)] |
| **[SRS\_BSW\_00007]** | All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard. | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00010]** | The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms. | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00101]** | The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function | [[SWS\_CANIF\_00001](#_bookmark256)] |
| **[SRS\_BSW\_00159]** | All modules of the AUTOSAR Basic Software shall support a tool based configuration | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00164]** | The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00167]** | All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00168]** | SW components shall be tested by a function defined in a common API in the Basis-SW | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00170]** | The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00172]** | The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00306]** | AUTOSAR Basic Software Modules shall be compiler and platform independent | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00307]** | Global variables naming convention | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00308]** | AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[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\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00312]** | Shared code shall be reentrant | [[SWS\_CANIF\_00064](#_bookmark193)] |
| **[SRS\_BSW\_00348]** | All AUTOSAR standard types and constants shall be placed and organized in a standard type header file | [[SWS\_CANIF\_00142](#_bookmark248)] |
| **[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](#_bookmark248)] |
| **[SRS\_BSW\_00358]** | The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void | [[SWS\_CANIF\_00001](#_bookmark256)] |
| **[SRS\_BSW\_00361]** | All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header | [[SWS\_CANIF\_00142](#_bookmark248)] |
| **[SRS\_BSW\_00373]** | The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00378]** | AUTOSAR shall provide a boolean type | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00405]** | BSW Modules shall support multiple configuration sets | [[SWS\_CANIF\_00001](#_bookmark256)] |
| **[SRS\_BSW\_00407]** | Each BSW module shall provide a function to read out the version information of a dedicated module implementation | [[SWS\_CANIF\_00158](#_bookmark313)] |
| **[SRS\_BSW\_00411]** | All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API | [[SWS\_CANIF\_00158](#_bookmark313)] |
| **[SRS\_BSW\_00414]** | Init functions shall have a pointer to a configuration structure as single parameter | [[SWS\_CANIF\_00001](#_bookmark256)] |
| **[SRS\_BSW\_00416]** | The sequence of modules to be initialized shall be configurable | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[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\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00423]** | BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00424]** | BSW module main processing functions shall not be allowed to enter a wait state | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00425]** | The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00426]** | BSW Modules shall ensure data consistency of data which is shared between BSW modules | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00427]** | ISR functions shall be defined and documented in the BSW module description template | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[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\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00429]** | Access to OS is restricted | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00432]** | Modules should have separate main processing functions for read/receive and write/transmit data path | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_BSW\_00433]** | Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_Can\_01001]** | The CAN Interface implementation and interface shall be independent from underlying CAN Controller and CAN Transceiver | [[SWS\_CANIF\_00023](#_bookmark143)] |
| **[SRS\_Can\_01003]** | The appropriate higher communication stack shall be notified by the CAN Interface about an occurred reception | [[SWS\_CANIF\_00012](#_bookmark423)] |
| **[SRS\_Can\_01005]** | The CAN Interface shall perform a check for correct DLC of received PDUs | [[SWS\_CANIF\_00026](#_bookmark221)] |
| **[SRS\_Can\_01008]** | The CAN Interface shall provide a transmission request service | [[SWS\_CANIF\_00005](#_bookmark278)] |
| **[SRS\_Can\_01009]** | The CAN Interface shall provide a transmission confirmation dispatcher | [[SWS\_CANIF\_00007](#_bookmark376)] |
| **[SRS\_Can\_01011]** | The CAN Interface shall provide a transmit buffer | [[SWS\_CANIF\_00068](#_bookmark169)] |
| **[SRS\_Can\_01014]** | The CAN State Manager shall offer a network configuration independent interface for upper layers | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[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](#_bookmark218)] |
| **[SRS\_Can\_01020]** | The TX-Buffer shall be statically configurable | [[SWS\_CANIF\_00063](#_bookmark165)] |
| **[SRS\_Can\_01021]** | CAN The CAN Interface shall implement an interface for initialization | [[SWS\_CANIF\_00001](#_bookmark256)] |
| **[SRS\_Can\_01022]** | The CAN Interface shall support the selection of configuration sets | [[SWS\_CANIF\_00001](#_bookmark256)] |
| **[SRS\_Can\_01027]** | The CAN Interface shall provide a service to change the CAN Controller mode. | [[SWS\_CANIF\_00003](#_bookmark263)] |
| **[SRS\_Can\_01028]** | The CAN Interface shall provide a service to query the CAN controller state | [[SWS\_CANIF\_00229](#_bookmark268)] |
| **[SRS\_Can\_01029]** | The CAN Interface shall report bus-off state of a device to an upper layer | [[SWS\_CANIF\_00014](#_bookmark426)] |
| **[SRS\_Can\_01114]** | Data Consistency of L-PDUs to transmit shall be guaranteed | [[SWS\_CANIF\_00033](#_bookmark177)] |
| **[SRS\_Can\_01125]** | The CAN stack shall ensure not to lose messages in receive direction | [[SWS\_CANIF\_00194](#_bookmark288)] |
| **[SRS\_Can\_01126]** | The CAN stack shall be able to produce 100% bus load | [[SWS\_CANIF\_00381](#_bookmark166)]  [[SWS\_CANIF\_00382](#_bookmark158)] [[SWS\_CANIF\_00881](#_bookmark167)] |
| **[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](#_bookmark288)] |
| **[SRS\_Can\_01130]** | Receive Status Interface of CAN Interface | [[SWS\_CANIF\_00202](#_bookmark293)]  [[SWS\_CANIF\_00230](#_bookmark298)] |
| **[SRS\_Can\_01131]** | The CAN Interface module shall provide the possibility to have polling and callback notification mechanism in parallel | [[SWS\_CANIF\_00230](#_bookmark298)] |
| **[SRS\_Can\_01136]** | The CAN Interface module shall provide a service to check for validation of a CAN wake-up event | [[SWS\_CANIF\_00179](#_bookmark205)] |
| **[SRS\_Can\_01139]** | The CAN Interface and Driver shall offer a CAN Controller specific interface for initialization | [[SWS\_CANIF\_00999](#_bookmark562)] |
| **[SRS\_Can\_01140]** | The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers | [[SWS\_CANIF\_00281](#_bookmark155)]  [[SWS\_CANIF\_00877](#_bookmark154)] |
| **[SRS\_Can\_01141]** | The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers at same time on one network | [[SWS\_CANIF\_00243](#_bookmark282)]  [[SWS\_CANIF\_00877](#_bookmark154)] |
| **[SRS\_Can\_01151]** | The CAN Interface shall provide a service to check for a CAN Wake-up event. | [[SWS\_CANIF\_00286](#_bookmark204)] |
| **[SRS\_Can\_01162]** | The CAN Interface shall support classic CAN and CAN FD frames | [[SWS\_CANIF\_00877](#_bookmark154)] |
| **[SRS\_Can\_01168]** | The CAN Interface shall implement an interface for de-initialization | [[SWS\_CANIF\_91002](#_bookmark260)] |
| **[SRS\_Can\_01169]** | The CAN interface shall provide a function to return the current CAN controller error state | [[SWS\_CANIF\_91001](#_bookmark273)] |
| **[SRS\_Can\_01172]** | The CAN Interface shall provide a function to provide received and transmitted frames to the Bus Mirroring | [[SWS\_CANIF\_00903](#_bookmark65)]  [[SWS\_CANIF\_00904](#_bookmark160)] [[SWS\_CANIF\_00905](#_bookmark179)] [[SWS\_CANIF\_00906](#_bookmark184)] [[SWS\_CANIF\_00911](#_bookmark362)] |
| **[SRS\_Can\_01181]** | The CAN Driver shall support hardware-based timestamping | [[SWS\_CANIF\_91011](#_bookmark367)]  [[SWS\_CANIF\_91012](#_bookmark369)] [[SWS\_CANIF\_91013](#_bookmark371)] [[SWS\_CANIF\_91014](#_bookmark365)] |

### Functional specification

#### General Functionality

The services of [CanIf](#_bookmark8)  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](#_bookmark8):

1. Interrupt Mode

[CanDrv](#_bookmark5) processes interrupts triggered by the [CAN Controller](#_bookmark24). [CanIf](#_bookmark8), which is event based, is notified when an event occurs. In this case the relevant [CanIf](#_bookmark8) services are called within the corresponding *ISRs* in [CanDrv](#_bookmark5).

1. Polling Mode

[CanDrv](#_bookmark5) is triggered by the [SchM](#_bookmark23) 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](#_bookmark8) is notified by [CanDrv](#_bookmark5) about events (*Reception*, *Transmission*, *BusOff*, *Timeout* ), that occurred in one of the [CAN Controllers](#_bookmark24), equally to the interrupt driven operation. [CanDrv](#_bookmark5) is responsible for the update of the corresponding infor- mation which belongs to the occurred event in the [CAN Controller](#_bookmark24), for example reception of a [L-PDU](#_bookmark3).

1. Mixed Mode: interrupt and polling driven [CanDrv](#_bookmark5)

The functionality can be divided between *interrupt driven* and *polling driven* opera- tion mode depending on the used [CAN Controllers](#_bookmark24).

Examples: Polling driven *FullCAN* reception and interrupt driven *BasicCAN* recep-

tion, 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](#_bookmark8) 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](#_bookmark8).

#### Hardware object handles

[Hardware Object Handles](#_bookmark17) ([HOH](#_bookmark17)) for transmission ([HTH](#_bookmark19)) as well as for reception ( [HRH](#_bookmark18)) 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](#_bookmark29) is referenced in [CanIf](#_bookmark8) independent of the CAN hardware buffer layout. The [HOH](#_bookmark17) is used as a parameter in the calls of [CanDrv](#_bookmark5)’s interface services and is provided by [CanDrv](#_bookmark5)’s configuration and used by [CanDrv](#_bookmark5) as identifier for communication buffers of the CAN mailbox.

[CanIf](#_bookmark8) acts only as user of the [Hardware Object Handle](#_bookmark17), but does not interpret it on the basis of hardware specific information. [CanIf](#_bookmark8) therefore remains independent of hardware.

**[SWS\_CANIF\_00023]** [[CanIf](#_bookmark8) shall avoid direct access to hardware specific commu- nication buffers and shall access it exclusively via [CanDrv](#_bookmark5) interface services.]*(*[*SRS\_-*](#_bookmark110)[*Can\_01001*](#_bookmark110)*)*

Rationale for [[SWS\_CANIF\_00023](#_bookmark143)]: [CanIf](#_bookmark8) remains independent of hardware, be- cause [CanDrv](#_bookmark5) interfaces are called with [HOH](#_bookmark17) parameters, which abstract from the concrete CAN hardware buffer properties.

Each [CAN Controller](#_bookmark24) can provide multiple [CAN Transmit Hardware Objects](#_bookmark29) in the CAN mailbox. These can be logically linked to one entire pool of [Hardware](#_bookmark29) [Objects](#_bookmark29) (multiplexed [Hardware Objects](#_bookmark29)) and thus addressed by one [HTH](#_bookmark19).

**[SWS\_CANIF\_00662]** *[*[CanIf](#_bookmark8) shall use two types of [HOHs](#_bookmark17) to enable access to [Can-](#_bookmark5) [Drv](#_bookmark5):

* [Hardware Transmit Handle](#_bookmark31) ([HTH](#_bookmark19)) and
* [Hardware Receive Handle](#_bookmark30) ([HRH](#_bookmark18)).

**[SWS\_CANIF\_00291]** [Definition of [HRH](#_bookmark18): The [HRH](#_bookmark18) shall be a handle referencing a logical [Hardware Receive Object](#_bookmark29) of the CAN Controller mailbox.*]()*

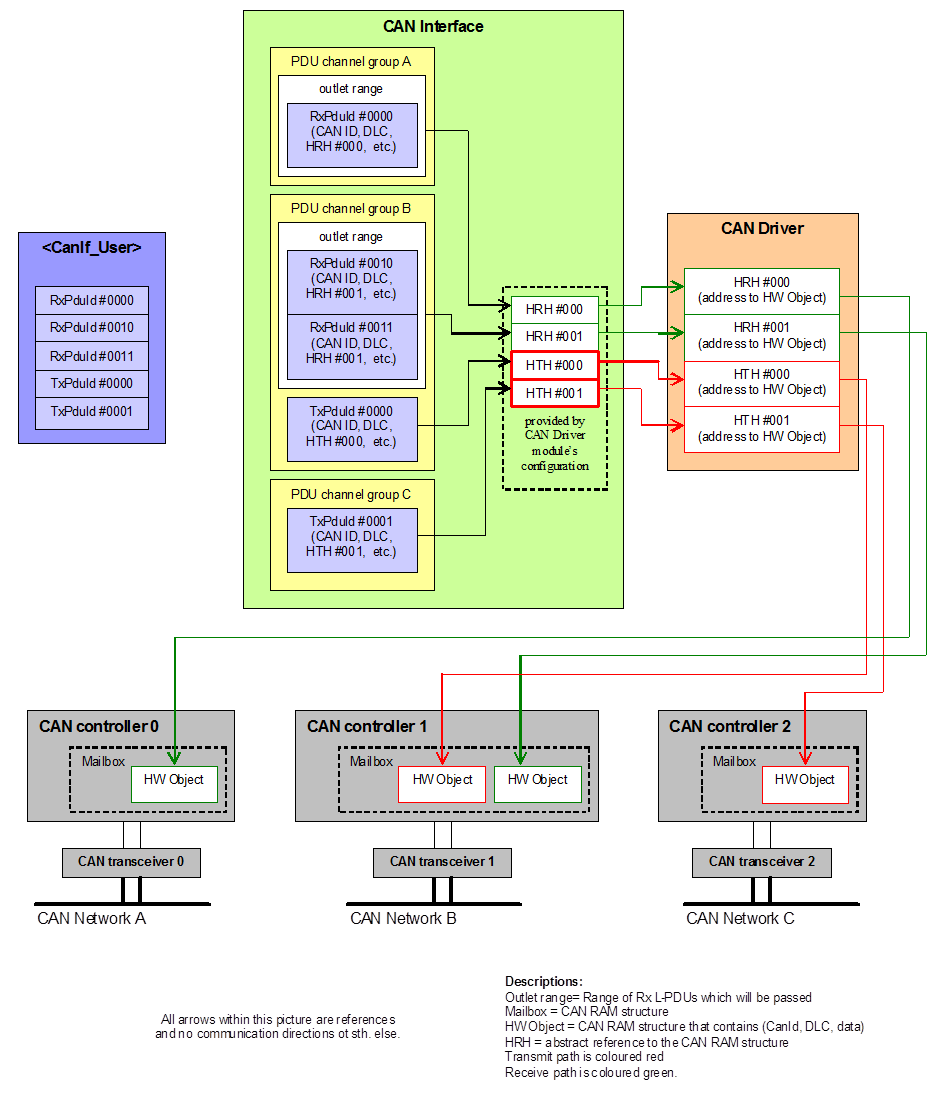
**[SWS\_CANIF\_00665]** [The [HRH](#_bookmark18) shall enable [CanIf](#_bookmark8) to use *BasicCAN* or a *FullCAN* reception method of the referenced reception unit and to indicate a Received [L-SDU](#_bookmark4) to a target upper layer module.*]()*

**[SWS\_CANIF\_00663]** *[*If the [HRH](#_bookmark18) references a reception unit configured for *BasicCAN reception*, software filtering shall be enabled in [CanIf](#_bookmark8).*]()*

**[SWS\_CANIF\_00664]** *[*If multiple [HRHs](#_bookmark18) are used, each [HRH](#_bookmark18) shall belong at least to a single or fixed group of Rx [L-SDU](#_bookmark4) (CanRxPduIds).*]()*

The [HRH](#_bookmark18) can be configured to receive

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



**Figure 7.1: Mapping between PDU Ids and HW object handles**

**[SWS\_CANIF\_00292]** *[*Definition of [HTH](#_bookmark19): The [HTH](#_bookmark19) shall be a handle referencing a logical [Hardware Transmit Object](#_bookmark29) of the CAN Controller mailbox.*]()*

**[SWS\_CANIF\_00666]** *[*The [HTH](#_bookmark19) shall enable [CanIf](#_bookmark8) to use *BasicCAN* or *FullCAN* transmission method of the referenced transmission unit and to confirm a transmitted [L-SDU](#_bookmark4) to a target upper layer module.*]()*

**[SWS\_CANIF\_00466]** *[*Each [CanIf Tx L-PDU](#_bookmark27) shall statically be assigned to one [CanIfBufferCfg](#_bookmark556) configuration container at configuration time (see [CanIfTxP-](#_bookmark493) [duBufferRef](#_bookmark493)).*]()*

Rationale for [[SWS\_CANIF\_00466](#_bookmark144)]: [CanIf Tx L-PDUs](#_bookmark27) do not refer [HTHs](#_bookmark19), but [Can-](#_bookmark556) [IfBufferCfg](#_bookmark556), which in turn do refer [HTHs](#_bookmark19).

**[SWS\_CANIF\_00667]** *[*If multiple [HTHs](#_bookmark19) are used, each [HTH](#_bookmark19) shall belong to a single or fixed group of [Tx L-PDU](#_bookmark27) (CanTxPduIds).*]()*

**[SWS\_CANIF\_00115]** *[*[CanIf](#_bookmark8) shall be able to use all [HRHs](#_bookmark18) and [HTHs](#_bookmark19) of one [CanDrv](#_bookmark5)

as common, single numbering area starting with zero.*]()*

The dedicated [HRHs](#_bookmark18) and [HTHs](#_bookmark19) are derived from the configuration set of [CanDrv](#_bookmark5). The definition of [HTH](#_bookmark19)/[HRH](#_bookmark18) inside the numbering area and [Hardware Objects](#_bookmark29) is up to [CanDrv](#_bookmark5).

#### Static L-PDUs

[CanIf](#_bookmark8) offers general access to the [CAN L-SDU](#_bookmark4) related data for upper layers. At- tributes of the following table are represented as configuration parameters and are specified in [chapter](#_bookmark451) [10](#_bookmark451):

|  |  |
| --- | --- |
| **CAN Interface specific attributes** | **CAN Protocol Control Information (PCI)** |
| Method of SW filtering  [CanIfPrivateSoftwareFilterType](#_bookmark459) | [CAN Identifier](#_bookmark7) ([CanId](#_bookmark7))  [CanIfTxPduCanId](#_bookmark482), range of [CanIds](#_bookmark7) per *PDU*  (see [CanIfRxPduCanIdRange](#_bookmark507)),  CanIfRxPduCanId, [CanIfRxPduCanIdMask](#_bookmark497) |
| Direction of [L-PDU](#_bookmark3) (Tx, Rx) [CanIfTxPduId](#_bookmark485),  [CanIfRxPduId](#_bookmark501)) | Type of [CAN Identifier](#_bookmark7) (*StandardCAN*,  *ExtendedCAN*) referenced from [CanDrv](#_bookmark5) via  [CanIfHthIdSymRef](#_bookmark542), [CanIfHrhIdSymRef](#_bookmark547) |
| [HTH](#_bookmark19)/[HRH](#_bookmark18) of the [CAN Controller](#_bookmark24) | Data Length and Data Length Code (*DLC*)  [CanIfRxPduDataLength](#_bookmark499) |
| Target ID for the corresponding upper layer  [CanIfTxPduUserTxConfirmationUL](#_bookmark492), [CanIfRxPduUserRxIndicationUL](#_bookmark505) | Reference to the PDU data (see [[1](#_bookmark36),  Specification of CAN Driver]) |
| Type of [Transmit L-PDU](#_bookmark27) (STATIC, DYNAMIC)  [CanIfTxPduType](#_bookmark489) |  |
| Type of [Tx/Rx L-PDU](#_bookmark26) (*FullCAN*, *BasicCAN*)  [CanIfHthIdSymRef](#_bookmark542), [CanIfHrhIdSymRef](#_bookmark547) |  |

[CanIf](#_bookmark8) supports activation and deactivation of all [L-PDUs](#_bookmark3) belonging to one [CAN Con-](#_bookmark24) [troller](#_bookmark24) for transmission as well as for reception (see [7.19.2](#_bookmark209), see [CanIf\_SetPdu-](#_bookmark304) [Mode()](#_bookmark304), [[SWS\_CANIF\_00008](#_bookmark303)]). For [L-PDU](#_bookmark3) mode control refer to [section](#_bookmark206) [7.19](#_bookmark206).

Each [L-PDU](#_bookmark3) is associated with an upper layer module in order to ensure correct dis- patching during reception, transmission confirmation, and data access. Each upper layer module can use the [L-PDUs](#_bookmark3) to serve different [CAN Controllers](#_bookmark24) simultane- ously.

According to the *PDU* architecture defined for the entire AUTOSAR communication stack (see [[7](#_bookmark42), Layered Software Architecture]), the usage of [L-PDUs](#_bookmark3) is split in two different ways:

* For transmission request and transmission/reception polling API the upper layer module uses the [L-SDU](#_bookmark4) ID (CanTxPduId/CanRxPduId) defined by [CanIf](#_bookmark8) as parameter.
* For all callback APIs, which are invoked by [CanIf](#_bookmark8) at upper layer modules, [CanIf](#_bookmark8) 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](#_bookmark3)/[L-SDU](#_bookmark4) Id of the callee.

If power on initialization is not performed and upper layer performs transmit requests to [CanIf](#_bookmark8), no [L-SDUs](#_bookmark4) 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](#_bookmark3)/[L-SDU](#_bookmark4) transmitting function is specified in detail in [subsection](#_bookmark277) [8.3.6](#_bookmark277).

#### Dynamic L-PDUs

[CanIf](#_bookmark8) shall support the ability to filter incoming messages using the CanIfRxPdu- CanIdMask. The filtering shall be done by comparing the incoming [CanId](#_bookmark7) with the stored [CanIfRxPduCanId](#_bookmark496) after applying the [CanIfRxPduCanIdMask](#_bookmark497) to both IDs. This should be done after the filtering of regular [CanIds](#_bookmark7) without mask, to allow for separate handling of some of the [CanIds](#_bookmark7) that fall into the range defined by the mask or a [CanId](#_bookmark7) based range.

Additionally, DYNAMIC Tx and Rx [L-SDUs](#_bookmark4) shall be supported, where the [CanId](#_bookmark7) resides in the MetaData of the [L-SDU](#_bookmark4).

During transmission of dynamic [L-SDUs](#_bookmark4), when a [CanIfTxPduCanIdMask](#_bookmark483) is defined, the variable parts of the [CanId](#_bookmark7) provided via the MetaData must be merged with the [CanId](#_bookmark7) by using this mask. When no [CanIfTxPduCanIdMask](#_bookmark483) and no [CanIfTxPdu-](#_bookmark482) [CanId](#_bookmark482) are configured, the MetaData shall be used directly as [CanId](#_bookmark7).

During reception of dynamic [L-SDUs](#_bookmark4), the received [CanId](#_bookmark7) shall be placed in the [L-SDU](#_bookmark4) MetaData. The content of the MetaData is independent of the [CanIfRxPduCanId-](#_bookmark497) [Mask](#_bookmark497) parameter.

**[SWS\_CANIF\_00844]** *[*[CanIf](#_bookmark8) shall support dynamic [L-PDUs](#_bookmark3), where the [CanId](#_bookmark7) or relevant parts of the [CanId](#_bookmark7) are placed in the MetaData of a [L-SDU](#_bookmark4).*]()*

##### Dynamic Transmit L-PDUs

Definition of dynamic [Transmit L-PDUs](#_bookmark27): [L-PDUs](#_bookmark3) which allow reconfiguration of the [CanId](#_bookmark7) during runtime ([CanIfTxPduType](#_bookmark489)) or where the ID or parts thereof are pro- vided as MetaData of the [L-SDU](#_bookmark4).

The usage of all other [L-PDU](#_bookmark3) elements are equal to normal static [Transmit L-PDUs](#_bookmark27):

* The transmit confirmation notification [CanIfTxPduUserTxConfirmationUL](#_bookmark492)

cannot be reconfigured as it belongs to the [L-PDU](#_bookmark3).

* The Data Length and the pointer to the data buffer are both determined by the upper layer module at call of [CanIf\_Transmit()](#_bookmark279).

The function [CanIf\_SetDynamicTxId()](#_bookmark317) (see [[SWS\_CANIF\_00189](#_bookmark316)]) reconfigures the [CanId](#_bookmark7) of a dynamic [L-PDU](#_bookmark3) with [CanIfTxPduType](#_bookmark489).

**[SWS\_CANIF\_00188]** *[*[CanIf](#_bookmark8) shall process the two most significant bits of the CanId (see [[1](#_bookmark36), 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](#_bookmark27) shall be transmitted.*]()*

**[SWS\_CANIF\_00673]** *[*The CanIf shall guarantee data consistency of the [CanId](#_bookmark7) in case of running function [CanIf\_SetDynamicTxId()](#_bookmark317). This service may be in- terrupted by a *pre-emptive* call of [CanIf\_Transmit()](#_bookmark279) affecting the same L-PDU, see [[SWS\_CANIF\_00064](#_bookmark193)].*]()*

**[SWS\_CANIF\_00855]** *[*If CanIfTxPduCanIdMask and CanIfTxPduCanId are omit- ted, the [CanId](#_bookmark7) 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](#_bookmark4).*]()*

**[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 CanIfTxPdu- CanIdType that shall appear in the actual [CanId](#_bookmark7), the other bits are taken from the MetaData.*]()*

Note: The resulting ID could be calculated in the following way: (CanIfTxPduCanId &

CanIfTxPduCanIdMask) *|* (<dynamic ID parts> & *∼*CanIfTxPduCanIdMask)

**[SWS\_CANIF\_00857]** *[*[CanIf\_Init()](#_bookmark257) (see [[SWS\_CANIF\_00085](#_bookmark258)]) initializes the [CanIds](#_bookmark7) of the dynamic [Transmit L-PDUs](#_bookmark27) with [CanIfTxPduType](#_bookmark489) to the value con- figured via CanIfTxPduCanId.*]()*

##### Dynamic receive L-PDUs

Definition of dynamic [Receive L-PDUs](#_bookmark26): L-PDUs that correspond to a set of [CanIds](#_bookmark7), where the actually received [CanId](#_bookmark7) is provided to upper layers as part of the PDU data.

**[SWS\_CANIF\_00847]** *[*Configuration shall ensure that dynamic [Receive L-PDUs](#_bookmark26) use an ID range or a mask and that the MetaDataItem CAN\_ID\_32 is configured for the [L-SDU](#_bookmark4). Besides, the software filtering must be enabled for these [L-SDUs](#_bookmark4).*]()*

**[SWS\_CANIF\_00848]** *[*Upon reception of a dynamic [L-SDU](#_bookmark4), [CanIf](#_bookmark8) shall place the

[CanId](#_bookmark7) in the MetaDataItem of type CAN\_ID\_32.*]()*

#### Physical channel view

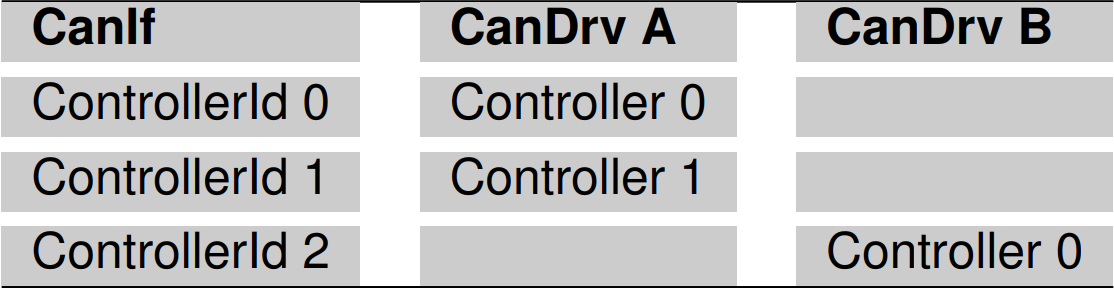
A physical channel is linked with one CAN Controller and one CAN Transceiver, whereas one or multiple physical channels may be connected to a single network.

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](#_bookmark14) (see [[3](#_bookmark38)]) 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](#_bookmark426)]).

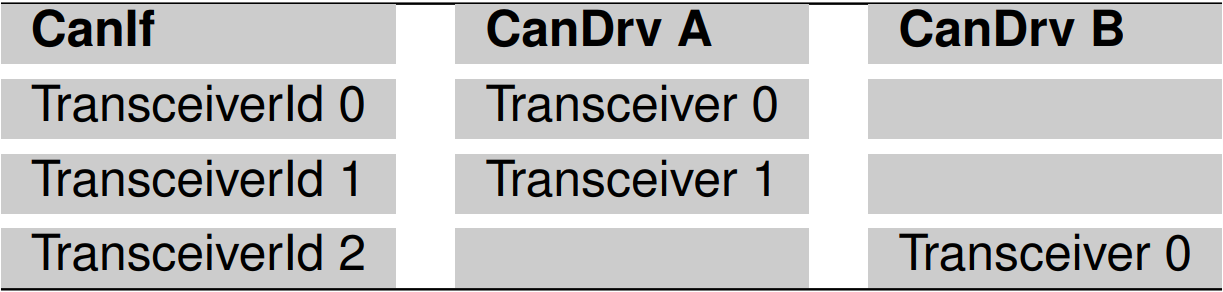
**[SWS\_CANIF\_00653]** *[*The CanIf shall provide a ControllerId, which abstracts from the different Controllers of the different CanDrv instances. The range of the Con- trollerIds within the CanIf shall start with ’0’. It shall be configurable via [CanIfC-](#_bookmark526) [trlId](#_bookmark526).*]()*

Example:

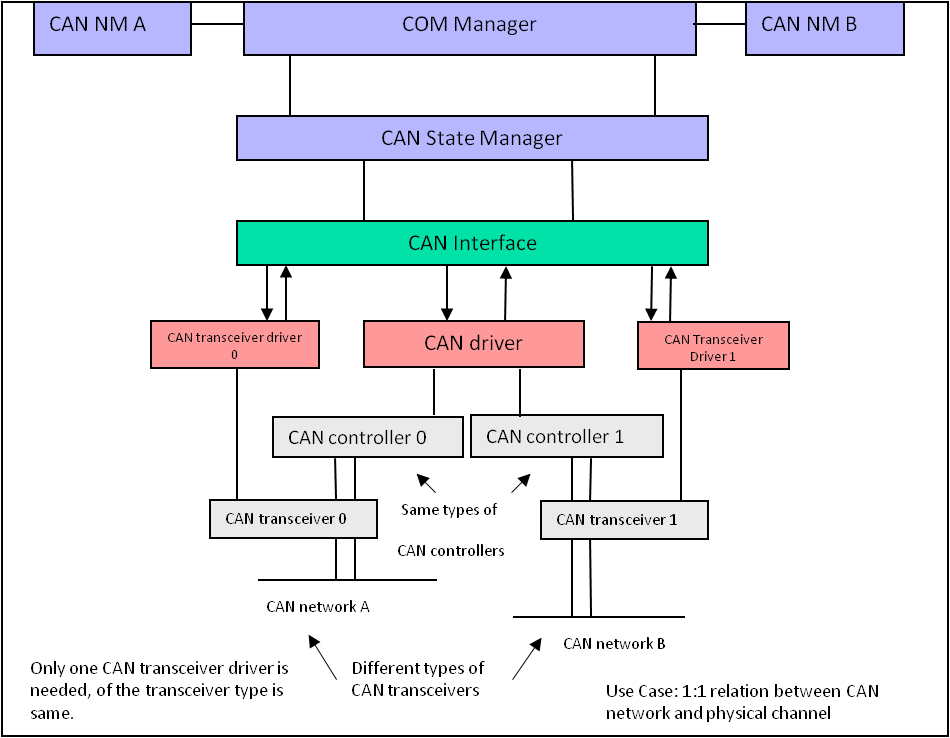


**[SWS\_CANIF\_00655]** *[*The CanIf shall provide a TransceiverId, which abstracts from the different Transceivers of the different CanTrcv instances. The range of the TransceiverIds within the CanIf shall start with ’0’. It shall be configurable via [CanIfTrcvId](#_bookmark535).*]()*

Example:



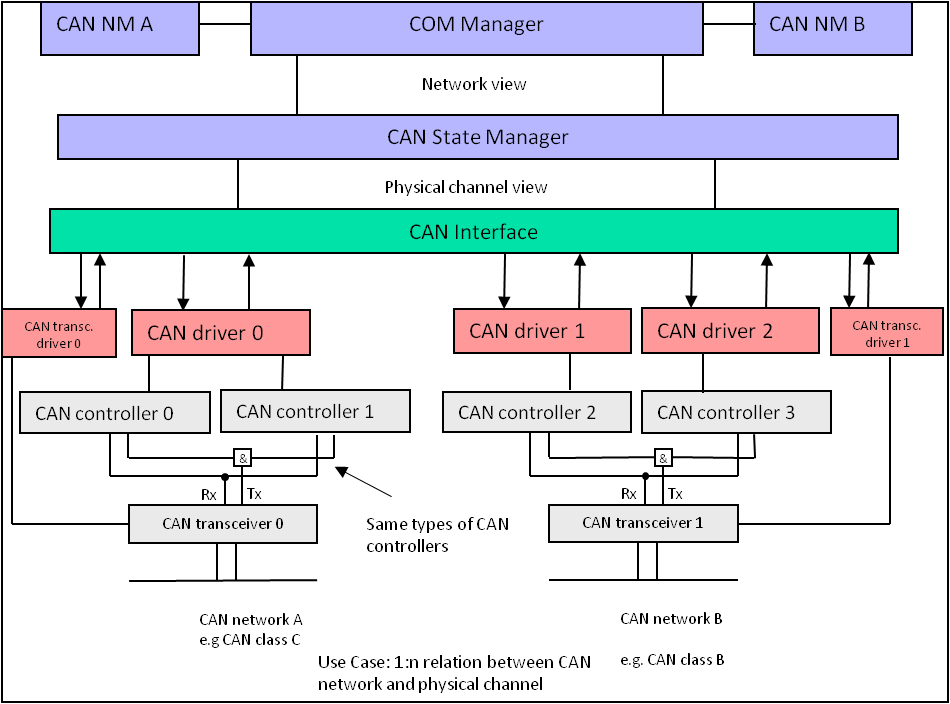
During the notification process the CanIf maps the original CAN Controller or CAN Transceiver parameter from the Driver module to the CanSm. This mapping is done as the referenced CAN Controller or CAN Transceiver parameters are configured with the abstracted CanIf parameters ControllerId or TransceiverId.



**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 Con- trollers are connected.



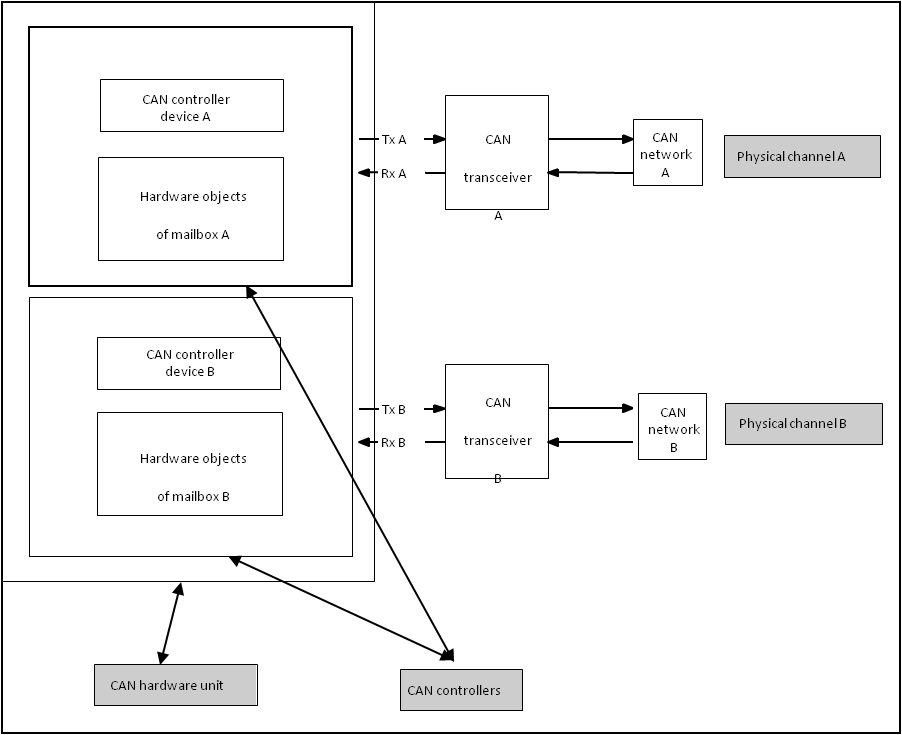
**Figure 7.3: Physical channel view definition example B**

#### 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](#_bookmark5).

If different types of [CAN Controllers](#_bookmark24) are used, also different types of [CanDrvs](#_bookmark5) have to be applied with a unified API to [CanIf](#_bookmark8). [CanIf](#_bookmark8) collects information about number and types of [CAN Controllers](#_bookmark24) and their [Hardware Objects](#_bookmark29) at configuration time. This allows transparent and hardware independent access to the [CAN Controllers](#_bookmark24) from upper layer modules using [HOHs](#_bookmark17) (refer to [section](#_bookmark142) [7.2](#_bookmark142) “[Hardware object handles](#_bookmark142)” and [section](#_bookmark225) [7.24](#_bookmark225) “[Multiple CAN Driver support](#_bookmark225)”).

[Figure](#_bookmark151) [7.4](#_bookmark151) 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**

#### BasicCAN and FullCAN reception

[CanIf](#_bookmark8) distinguishes between *BasicCAN* and *FullCAN* handling for activation of soft- ware acceptance filtering.

A CAN mailbox ([Hardware Object](#_bookmark29)) for *FullCAN* operation only enables transmission or reception of single [CanIds](#_bookmark7). Accordingly, *BasicCAN* operation of one [Hardware](#_bookmark29) [Object](#_bookmark29) enables to transmit or receive a range of [CanIds](#_bookmark7).

A [Hardware Receive Object](#_bookmark29) for configured *BasicCAN* reception is able to receive a range of [CanIds](#_bookmark7), which pass its hardware acceptance filter. This range may exceed the list of predefined [Rx L-PDUs](#_bookmark26) to be received by this [HRH](#_bookmark18). Therefore, [CanIf](#_bookmark8) subse- quently shall execute software filtering to pass only the predefined list of [Rx L-PDUs](#_bookmark26) to the corresponding upper layer modules. For more details please refer to [section](#_bookmark215) [7.20](#_bookmark215) “[Software receive filter](#_bookmark215)”.

**[SWS\_CANIF\_00467]** *[*[CanIf](#_bookmark8) shall configure and store an order on [HTHs](#_bookmark19) and [HRHs](#_bookmark18) for all [HOHs](#_bookmark17) derived from the configuration containers [CanIfHthCfg](#_bookmark540) and [Can-](#_bookmark544) [IfHrhCfg](#_bookmark544).*]()*

**[SWS\_CANIF\_00468]** *[*[CanIf](#_bookmark8) shall reference a hardware acceptance filter for each [HOH](#_bookmark17) derived from the configuration parameters [CanIfHthIdSymRef](#_bookmark542) and [Can-](#_bookmark547) [IfHrhIdSymRef](#_bookmark547).*]()*

The main difference between *BasicCAN* and *FullCAN* operation is in the need of a software acceptance filtering mechanism (see [section](#_bookmark215) [7.20](#_bookmark215) “[Software receive filter](#_bookmark215)”).

**[SWS\_CANIF\_00469]** *[*[CanIf](#_bookmark8) shall give the possibility to configure and store a soft- ware acceptance filter for each [HRH](#_bookmark18) of type *BasicCAN* configured by parameter [Can-](#_bookmark545) [IfHrhSoftwareFilter](#_bookmark545).*]()*

**[SWS\_CANIF\_00211]** *[*[CanIf](#_bookmark8) shall execute the software acceptance filter from [[SWS\_CANIF\_00469](#_bookmark153)] for the [HRH](#_bookmark18) passed by callback function [CanIf\_RxIndica-](#_bookmark383) [tion()](#_bookmark383).*]()*

*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](#_bookmark24) [Controllers](#_bookmark24).

**[SWS\_CANIF\_00877]** *[*If [CanIf](#_bookmark8) receives a [L-PDU](#_bookmark3) (see [CanIf\_RxIndication()](#_bookmark383)), it shall perform the following comparisons to select the correct reception [L-SDU](#_bookmark4) configured in CanIfRxPduCfg:

* compare [CanIfRxPduCanId](#_bookmark496) with the passed Mailbox->CanId (Can\_Id- Type) excluding the two most significant bits
* compare CanIfRxPduCanIdType with the two most significant bits of the passed Mailbox->CanId (Can\_IdType)

*]([SRS\_Can\_01140](#_bookmark132),* [*SRS\_Can\_01141*](#_bookmark133)*,* [*SRS\_Can\_01162*](#_bookmark135)*)*

Basically, [CanIf](#_bookmark8) supports reception either of *Standard CAN IDs* or *Extended CAN IDs* on one [Physical CAN Channel](#_bookmark32) by the parameters [CanIfTxPduCanIdType](#_bookmark484) and [CanIfRxPduCanIdType](#_bookmark498).

**[SWS\_CANIF\_00281]** *[*[CanIf](#_bookmark8) shall accept and handle *StandardCAN IDs* and *Ex- tendedCAN IDs* on the same [Physical Channel](#_bookmark32) (= mixed mode operation).*](*[*SRS\_-*](#_bookmark132)[*Can\_01140*](#_bookmark132)*)*

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](#_bookmark29) have been configured separately for either *StandardCAN* or *ExtendedCAN* operation using configuration pa- rameters [CanIfTxPduCanIdType](#_bookmark484) and [CanIfRxPduCanIdType](#_bookmark498). In case of mixed mode operation the software acceptance filter algorithm (see [section](#_bookmark215) [7.20](#_bookmark215) “[Software](#_bookmark215) [receive filter](#_bookmark215)”) must be able to deal with both type of [CanIds](#_bookmark7).

[[SWS\_CANIF\_00281](#_bookmark155)] is an optional feature. This feature can be realized by different variants of implementations, no configuration options are available.

#### Initialization

The [EcuM](#_bookmark16) calls the CanIf’s function [CanIf\_Init()](#_bookmark257) for initialization of the entire CanIf (see [[SWS\_CANIF\_00001](#_bookmark256)]). All global variables and data structures are initialized including flags and buffers during the initialization process. The EcuM executes initial- ization of [CanDrvs](#_bookmark5) and CanTrcvs separately by call of their corresponding initialization services (refer to [[1](#_bookmark36)] and [[2](#_bookmark37), Specification of CAN Transceiver Driver]).

The CanIf expects that the CAN Controller remains in *STOPPED* mode like af- ter 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](#_bookmark3) (see [[SWS\_CANIF\_00001](#_bookmark256)]).

If re-initialization of the entire CAN modules during runtime is required, the [EcuM](#_bookmark16) shall invoke the CanSm (see [[3](#_bookmark38)]) to initiate the required state transitions of the CAN Con- troller by call of CAN Interface module’s API service [CanIf\_SetControllerMode](#_bookmark264) [()](#_bookmark264). The CanIf maps the calls from CanSm to calls of the respective [CanDrvs](#_bookmark5) (see [subsection](#_bookmark418) [8.6.3](#_bookmark418)).

#### Transmit request

[CanIf](#_bookmark8)’s transmit request function [CanIf\_Transmit()](#_bookmark279) ([[SWS\_CANIF\_00005](#_bookmark278)]) is a common interface for upper layers to transmit [L-PDUs](#_bookmark3) on the CAN network. The up- per communication layer modules initiate the transmission only via [CanIf](#_bookmark8)’s services without direct access to [CanDrv](#_bookmark5). The initiated [Transmit Request](#_bookmark33) is successfully completed, if [CanDrv](#_bookmark5) could write the [L-PDU](#_bookmark3) data into the CAN hardware transmit ob- ject.

Upper layer modules use the API service [CanIf\_Transmit()](#_bookmark279) to initiate a transmit request (refer to [subsection](#_bookmark277) [8.3.6](#_bookmark277) “[CanIf\_Transmit](#_bookmark277)”).

[CanIf](#_bookmark8) performs following actions for [L-PDU](#_bookmark3) transmission at call of the service [CanIf\_Transmit()](#_bookmark279):

* Check, initialization status of [CanIf](#_bookmark8)
* Identify [CanDrv](#_bookmark5) (only if multiple [CanDrvs](#_bookmark5) are used)
* Determine [HTH](#_bookmark19) for access to the CAN hardware transmit object
* Call Can\_Write() of [CanDrv](#_bookmark5)

The transmission is successfully completed, if the transmit request service [CanIf\_-](#_bookmark279) [Transmit()](#_bookmark279) 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()](#_bookmark279) shall return E\_NOT\_OK.*](*[*SRS\_Can\_01126*](#_bookmark126)*)*

Note for [[SWS\_CANIF\_00382](#_bookmark158)]: See [subsection](#_bookmark209) [7.19.2](#_bookmark209) “[PDU channel modes](#_bookmark209)”.

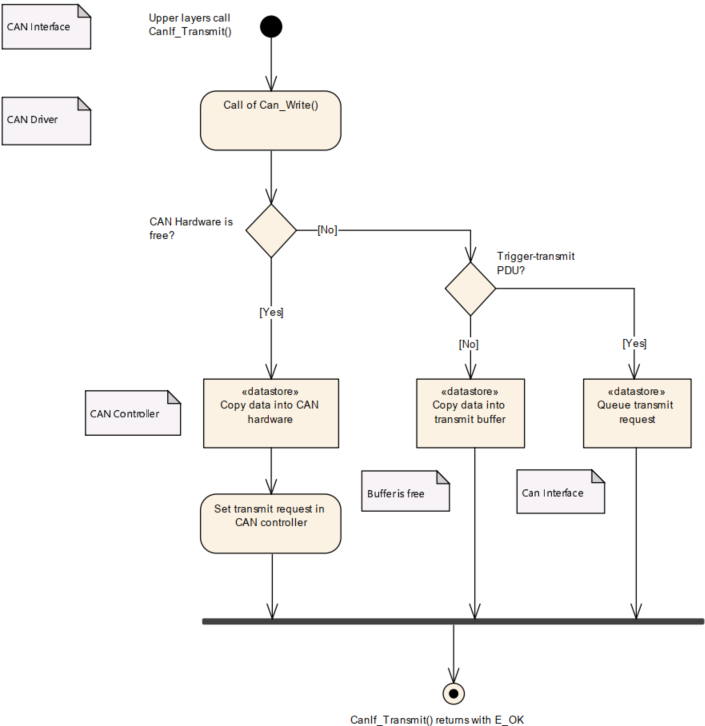
If the call of Can\_Write() returns with CAN\_BUSY, please refer to [section](#_bookmark178) [7.12](#_bookmark178) “[Trans-](#_bookmark178) [mit confirmation](#_bookmark178)” for further details.

#### Transmit data flow

The [Transmit Request](#_bookmark33) service [CanIf\_Transmit()](#_bookmark279) is based on [L-PDUs](#_bookmark3). The access to the [L-SDU](#_bookmark4) specific data is organized by the following parameters:

* [Transmit L-PDU](#_bookmark27) => [L-SDU](#_bookmark4) ID
* Reference to a data structure, which contains [L-SDU](#_bookmark4) related data: Pointer to the [L-SDU](#_bookmark4), pointer to the MetaData and [L-SDU](#_bookmark4) length.

The reference to the [L-SDU](#_bookmark4) data structure is used as a parameter in several [CanIf](#_bookmark8)’s API services, e.g. [CanIf\_Transmit()](#_bookmark279) or the callback service <User\_RxIndica- tion>(). In case the [L-PDU](#_bookmark3) is configured for triggered transmission, the [L-SDU](#_bookmark4) pointer is a null pointer.

****

**Figure 7.5: Transmit data flow**

[CanIf](#_bookmark8) stores information about the available hardware objects configured for trans- mission purposes. The function [CanIf\_Transmit()](#_bookmark279) maps the CanTxPduId to the corresponding [HTH](#_bookmark19) and calls the function Can\_Write() (see [[SWS\_CANIF\_00318](#_bookmark281)]).

**[SWS\_CANIF\_00904]** *[*If Bus Mirroring is enabled globally (see [CanIfBusMirror-](#_bookmark462) [ingSupport](#_bookmark462)) and has been activated with a call to [CanIf\_EnableBusMirroring](#_bookmark361) [()](#_bookmark361) for a [CAN Controller](#_bookmark24), the [CanIf](#_bookmark8) shall store the content of each frame before it is transmitted on that controller with Can\_Write().*](*[*SRS\_Can\_01172*](#_bookmark138)*)*

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()](#_bookmark377).

#### Transmit buffering

##### General behavior

At the scope of [CanIf](#_bookmark8) the transmit process starts with the call of [CanIf\_Transmit()](#_bookmark279) and it ends with invocation of upper layer module’s callback service <User\_TxCon- firmation>(). During the transmit process [CanIf](#_bookmark8), [CanDrv](#_bookmark5) and the CAN Mailbox altogether shall store the [L-PDU](#_bookmark3) to be transmitted only once at a single location. De- pending on the transmit method, these are:

* The CAN hardware transmit object or
* The [Transmit L-PDU Buffer](#_bookmark28) inside [CanIf](#_bookmark8), if transmit buffering is enabled. For triggered transmission, [CanIf](#_bookmark8) only has to store the transmit request for the given

[L-PDU](#_bookmark3) but not its data. The data is fetched just in time by means of the trigger transmit function when the [HTH](#_bookmark19) is free (again). A single [Tx L-PDU](#_bookmark27), 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](#_bookmark8) will store a [Tx L-PDU](#_bookmark27) in a [CanIf Trans-](#_bookmark28) [mit L-PDU Buffer](#_bookmark28) ([CanIfBufferCfg](#_bookmark556)), if it is rejected by [CanDrv](#_bookmark5) at [Transmit](#_bookmark33) [Request](#_bookmark33).

Basically, the overall buffer in [CanIf](#_bookmark8) for buffering [Tx L-PDUs](#_bookmark27) consits of one or multi- ple [CanIfBufferCfg](#_bookmark556) (see [CanIfBufferCfg](#_bookmark556)). Whereas each [CanIfBufferCfg](#_bookmark556) is assigned to one or multiple dedicated [CanIfBufferHthRef](#_bookmark558) (see [CanIfBuffer-](#_bookmark558) [HthRef](#_bookmark558)) and can be configured to buffer one or multiple [Tx L-PDUs](#_bookmark27). But as al- ready mentioned above only one instance per [Tx L-PDU](#_bookmark27) can be buffered in the overall amount of [CanIfBufferCfg](#_bookmark556).

The behavior of [CanIf](#_bookmark8) during [L-PDU](#_bookmark3) transmission differs whether transmit buffering is enabled in the configuration setup for the corresponding [Tx L-PDU](#_bookmark27), or not. If trans- mit buffering is disabled and a transmit request to [CanDrv](#_bookmark5) fails ([CAN Controller](#_bookmark24) mailbox is in use, *BasicCAN*), the [L-PDU](#_bookmark3) is not copied to the [CAN Controller](#_bookmark24)’s mailbox and [CanIf\_Transmit()](#_bookmark279) returns the value E\_NOT\_OK. If transmit buffering is enabled and a transmit request to [CanDrv](#_bookmark5) fails, depending on the CanIfTxBuffer configuration the [L-PDU](#_bookmark3) can be stored in a CanIfTxBuffer. In this case the API [CanIf\_Transmit()](#_bookmark279) returns the value E\_OK although the transmission could not be performed. In this case [CanIf](#_bookmark8) takes care of the outstanding transmission of the [L--PDU](#_bookmark3) via [CanIf\_TxConfirmation()](#_bookmark377) callback and the upper layer doesn’t have to retry the transmit request.

The number of available transmit [CanIf Tx L-PDU Buffers](#_bookmark28) can be configured completely independent from the number of used [Transmit L-PDUs](#_bookmark27) defined in the CAN network description file for this ECU.

As per [[SWS\_CANIF\_00835](#_bookmark168)] a [Tx L-PDU](#_bookmark27) refers [HTHs](#_bookmark19) via the [CanIfBufferCfg](#_bookmark556) con- figuration container (see [CanIfBufferCfg](#_bookmark556)). This is valid if transmit buffering is not needed as well. In this case, the buffer size (see [CanIfBufferSize](#_bookmark557)) of the [Can-](#_bookmark556) [IfBufferCfg](#_bookmark556) has to be set to 0. Then [CanIfBufferCfg](#_bookmark556) configuration container is only used to refer a [HTH](#_bookmark19).

##### Buffer characteristics

[CanIfTxPduBufferRef](#_bookmark493), [CanIfBufferCfg](#_bookmark556), [CanIfBufferHthRef](#_bookmark558) and [CanIf-](#_bookmark557) [BufferSize](#_bookmark557) describe the possible [CanIfBufferCfg](#_bookmark556) configurations.

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

[CanIf](#_bookmark8) tries to store a new [Transmit L-PDU](#_bookmark27) or its [Transmit Request](#_bookmark33) in the [Transmit L-PDU Buffer](#_bookmark28) only, if [CanDrv](#_bookmark5) return CAN\_BUSY during a call of Can\_- Write() (see [[SWS\_CANIF\_00381](#_bookmark166)]).

**[SWS\_CANIF\_00063]** *[*The [CanIf](#_bookmark8) shall support buffering of a CAN L-PDU for *Ba- sicCAN* transmission in the [CanIf](#_bookmark8), if parameter [CanIfPublicTxBuffering](#_bookmark471) (see [CanIfPublicTxBuffering](#_bookmark471)) is enabled.*](*[*SRS\_Can\_01020*](#_bookmark118)*)*

**[SWS\_CANIF\_00849]** *[*For dynamic [Transmit L-PDUs](#_bookmark27), also the CanId has to be stored in the CanIfTxBuffer.*]()*

**[SWS\_CANIF\_00381]** *[*If transmit buffering is enabled (see [[SWS\_CANIF\_00063](#_bookmark165)]) and if the call of Can\_Write() for a PDU configured for direct transmission returns with CAN\_BUSY, [CanIf](#_bookmark8) shall check if it is possible to buffer the [CanIf Tx L-PDU](#_bookmark27), which was requested to be transmitted via Can\_Write() in a CanIfTxBuffer.*] (*[*SRS\_Can\_01126*](#_bookmark126)*)*

When the call of Can\_Write() returns with CAN\_BUSY, [CanDrv](#_bookmark5) has rejected the requested transmission of the [L-PDU](#_bookmark3) (see [[1](#_bookmark36)]) because there is no free hardware object available at time of the transmit request ([Tx request](#_bookmark33)).

**[SWS\_CANIF\_00895]** *[*If the rejected data length exceeds the configured size, [CanIf](#_bookmark8)

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](#_bookmark165)]) and if the call of Can\_Write() for a PDU configured for triggered transmission returns with CAN\_BUSY, [CanIf](#_bookmark8) shall check if it is possible to buffer the [Transmit Request](#_bookmark33), which was requested to be transmitted via Can\_Write() in a CanIfTxBuffer.*] (*[*SRS\_Can\_01126*](#_bookmark126)*)*

**[SWS\_CANIF\_00835]** *[*When [CanIf](#_bookmark8) checks whether it is possible to buffer a [CanIf Tx L-PDU](#_bookmark27) or a [Transmit Request](#_bookmark33) (see [[SWS\_CANIF\_00381](#_bookmark166)], [[SWS\_CANIF\_00881](#_bookmark167)]), this shall only be possible, if the [CanIf Tx L-PDU](#_bookmark27) is as- signed (see [CanIfTxPduBufferRef](#_bookmark493)) to a [CanIfBufferCfg](#_bookmark556) (see [CanIfBuffer-](#_bookmark556) [Cfg](#_bookmark556)), which is configured with a buffer size (see [CanIfBufferSize](#_bookmark557)) 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](#_bookmark19) (see [CanIfBufferSize](#_bookmark557)).

**[SWS\_CANIF\_00836]** *[*If it is possible to buffer a [CanIf Tx L-PDU](#_bookmark27) or a [Transmit](#_bookmark33) [Request](#_bookmark33), because the buffer size of the assigned CanIfTxBuffer is bigger than zero (see [[SWS\_CANIF\_00835](#_bookmark168)]), [CanIf](#_bookmark8) shall buffer a [CanIf Tx L-PDU](#_bookmark27) or the [Transmit](#_bookmark33) [Request](#_bookmark33) in a free buffer element of the assigned CanIfTxBuffer, if the [CanIf Tx](#_bookmark27) [L-PDU](#_bookmark27) or the [Transmit Request](#_bookmark33) is not already buffered in the CanIfTxBuffer.*]()*

**[SWS\_CANIF\_00068]** *[*If it is possible to buffer a [CanIf Tx L-PDU](#_bookmark27) or a [Transmit](#_bookmark33) [Request](#_bookmark33), because the buffer size of the assigned CanIfTxBuffer is bigger than zero (see [[SWS\_CANIF\_00835](#_bookmark168)]), [CanIf](#_bookmark8) shall overwrite direct transmitted [CanIf Tx](#_bookmark27) [L-PDU](#_bookmark27) in the assigned CanIfTxBuffer, if the [CanIf Tx L-PDU](#_bookmark27) is already buffered in the CanIfTxBuffer when Can\_Write() returns CAN\_BUSY.*](*[*SRS\_Can\_01011*](#_bookmark115)*)*

Note: There is nothing to do for already stored [Transmit Requests](#_bookmark33) (see [[SWS\_CANIF\_00068](#_bookmark169)]) due to the fact the data will be catched by [CanDrv](#_bookmark5) di- rectly (using [CanIf\_TriggerTransmit()](#_bookmark374)). Therefore, the latest data will be sent automatically.

If the order of various transmit requests of different [L-PDUs](#_bookmark3) shall be kept, transmit requests of upper layer modules must be connected to previous transmit confirmation notifications. This means that a subsequent [L-PDU](#_bookmark3) is requested for transmission by the upper layer modules only, if the transmit confirmation of the previous one was notified by [CanIf](#_bookmark8).

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()](#_bookmark279) is called with a new [L-PDU](#_bookmark3) (no other instance of the same [L-PDU](#_bookmark3) is already stored in the buffer), then the new [L-PDU](#_bookmark3) or its [Transmit Request](#_bookmark33) shall not be stored and [CanIf\_Transmit()](#_bookmark279) shall return E\_NOT\_OK.*]()*

###### Clearance of transmit L-PDU buffers

**[SWS\_CANIF\_00386]** *[*[CanIf](#_bookmark8) shall evaluate during transmit confirmation (see [[SWS\_CANIF\_00007](#_bookmark376)]), whether pending [CanIf Tx L-PDUs](#_bookmark27) or [Transmit](#_bookmark33)

[Requests](#_bookmark33) are stored within the CanIfTxBuffers, which are assigned to the new free [Hardware Transmit Object](#_bookmark29) (see [[SWS\_CANIF\_00466](#_bookmark144)]).*]()*

**[SWS\_CANIF\_00668]** *[*If pending [CanIf Tx L-PDUs](#_bookmark27) or [Transmit Requests](#_bookmark33) are available in the CanIfTxBuffers as per [[SWS\_CANIF\_00386](#_bookmark171)], then [CanIf](#_bookmark8) shall call Can\_Write() for that pending [CanIf Tx L-PDU](#_bookmark27) or [Transmit Requests](#_bookmark33) (of the one assigned to the new [Hardware Transmit Object](#_bookmark29)) with the highest priority (see [[SWS\_CANIF\_00070](#_bookmark172)]).*]()*

**[SWS\_CANIF\_00070]** *[*[CanIf](#_bookmark8) shall transmit [L-PDUs](#_bookmark3) or [Transmit Requests](#_bookmark33) stored in the [Transmit L-PDU Buffers](#_bookmark28) in priority order (see [[12](#_bookmark47), ISO 11898-1:2015]) per each [HTH](#_bookmark19). [CanIf](#_bookmark8) shall not differentiate between [L-PDUs](#_bookmark3) and [Transmit Requests](#_bookmark33).*] ()*

**[SWS\_CANIF\_00183]** *[*When [CanIf](#_bookmark8) calls the function Can\_Write() for prioritized [L-PDUs](#_bookmark3) and [Transmit Requests](#_bookmark33) stored in CanIfTxBuffer and the return value of Can\_Write() is E\_OK, then [CanIf](#_bookmark8) shall remove this [L-PDU](#_bookmark3) or [Transmit Request](#_bookmark33) from the [Transmit L-PDU Buffer](#_bookmark28) immediately, before the transmit confirmation re- turns.*]()*

The behavior specified in [[SWS\_CANIF\_00183](#_bookmark173)] simplifies the choice of the new trans- mit [L-PDU](#_bookmark3) stored in the [Transmit L-PDU Buffer](#_bookmark28).

###### Initialization of transmit L-PDU buffers

**[SWS\_CANIF\_00387]** *[*When function [CanIf\_Init()](#_bookmark257) is called, [CanIf](#_bookmark8) shall initialize every [Transmit L-PDU Buffer](#_bookmark28) assigned to [CanIf](#_bookmark8).*]()*

The requirement [[SWS\_CANIF\_00387](#_bookmark175)] is necessary to prevent transmission of old data after restart of the [CAN Controller](#_bookmark24).

##### Data integrity of transmit L-PDU buffers

**[SWS\_CANIF\_00033]** *[*[CanIf](#_bookmark8) shall protect against concurrent access to [Trans-](#_bookmark28) [mit L-PDU Buffers](#_bookmark28) for transmit [L-PDUs](#_bookmark3) and [Transmit Requests](#_bookmark33).*]([SRS\_Can\_-](#_bookmark124)* [*01114*](#_bookmark124)*)*

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

Rationale: for [[SWS\_CANIF\_00033](#_bookmark177)]: pre-emptive accesses to the [Transmit L-PDU](#_bookmark28) [Buffer](#_bookmark28) cannot always be avoided. Such [Transmit L-PDU Buffer](#_bookmark28) access like storing a new [L-PDU](#_bookmark3) or removing transmitted [L-PDU](#_bookmark3) may occur preemptively.

#### Transmit confirmation

If a previous transmit request is completed successfully, [CanDrv](#_bookmark5) notifies it to [CanIf](#_bookmark8)

by the call of [CanIf\_TxConfirmation()](#_bookmark377) ([[SWS\_CANIF\_00007](#_bookmark376)]).

**[SWS\_CANIF\_00905]** *[*If Bus Mirroring is enabled globally (see [CanIfBusMirror-](#_bookmark462) [ingSupport](#_bookmark462)) and has been activated with a call to [CanIf\_EnableBusMirroring](#_bookmark361) [()](#_bookmark361) for a [CAN Controller](#_bookmark24), the [CanIf](#_bookmark8) shall call Mirror\_ReportCanFrame() for each frame transmission on that controller that is confirmed with [CanIf\_TxConfir-](#_bookmark377) [mation()](#_bookmark377), providing the stored content and the actual CAN ID.*](*[*SRS\_Can\_01172*](#_bookmark138)*)*

**[SWS\_CANIF\_00383]** *[*When callback notification [CanIf\_TxConfirma-](#_bookmark377) [tion()](#_bookmark377) is called, [CanIf](#_bookmark8) shall identify the upper layer communication layer (see [[SWS\_CANIF\_00414](#_bookmark380)]), which is linked to the successfully transmitted [L-](#_bookmark3)

[-PDU](#_bookmark3), and shall notify it about the performed transmission by call of [CanIf](#_bookmark8)’s transmit confirmation service <User\_TxConfirmation>(E\_OK).*]()*

Note for [[SWS\_CANIF\_00383](#_bookmark180)]: See [section](#_bookmark178) [7.12](#_bookmark178) “[Transmit confirmation](#_bookmark178)”.

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](#_bookmark3) or groups of [L-PDUs](#_bookmark3). All that services are called by [CanIf](#_bookmark8) 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 [subsubsection](#_bookmark420) [8.6.3.2](#_bookmark420) “[<User\_TxConfirmation>](#_bookmark420)”.

**[SWS\_CANIF\_00740]** *[*If [CanIfPublicTxConfirmPollingSupport](#_bookmark472) is enabled, [CanIf](#_bookmark8) shall buffer the information about a received TxConfirmation per [CAN Con-](#_bookmark24) [troller](#_bookmark24), if the controller mode of that controller is in state CAN\_CS\_STARTED.*]()*

#### Receive data flow

According to the AUTOSAR Basic Software Architecture the received data will be eval- uated and processed in the upper layer communication stacks (i.e. AUTOSAR COM, [CanNm](#_bookmark9), [CanTp](#_bookmark11), [DCM](#_bookmark15)). This means, upper layer modules may neither work with (i.e. change) buffers of [CanDrv](#_bookmark5) (Rx) nor do they have access to buffers of [CanIf](#_bookmark8) (Tx).

[CanIf](#_bookmark8) provides internal buffering in the receive path only if [CanIfPublicReadRxP-](#_bookmark467) [duDataApi](#_bookmark467) is set to TRUE (refer to [section](#_bookmark189) [7.15](#_bookmark189)). Tx buffering is addressed in [section](#_bookmark161)

[7.11](#_bookmark161) and dynamic [L-PDUs](#_bookmark3) are concerned in [section](#_bookmark146) [7.4](#_bookmark146).

In case of a new reception of an [L-PDU](#_bookmark3) [CanDrv](#_bookmark5) calls [CanIf\_RxIndication()](#_bookmark383) (refer to [[SWS\_CANIF\_00006](#_bookmark382)]) of [CanIf](#_bookmark8). The access to the [L-PDU](#_bookmark3) specific data is orga- nized by these parameters:

* Hardware Receive Handle ([HRH](#_bookmark18))
* Received CAN Identifier (CanId)
* Received Data Length
* Reference to [Received L-PDU](#_bookmark26)

The [Received L-PDU](#_bookmark26) is hardware dependent (nibble and byte ordering, access type) and allocated to the lowest layer in the communication system - to [CanDrv](#_bookmark5). [HRH](#_bookmark18) serves as a link between [CanDrv](#_bookmark5) and the upper layer module using the [L-PDU](#_bookmark3). The [HRH](#_bookmark18) identifies one CAN hardware receive object, where a new [CAN L-PDU](#_bookmark3) was received.

After the indication of a received [L-PDU](#_bookmark3) by [CanDrv](#_bookmark5) ([CanIf\_RxIndication()](#_bookmark383) is called) the [CanIf](#_bookmark8) shall proceed as described in [7.14](#_bookmark183) [Receive indication](#_bookmark183). [CanIf](#_bookmark8) is not able to recognize, whether [CanDrv](#_bookmark5) uses temporary buffering or a direct hardware access. It expects normalized [L-PDU](#_bookmark3) data in calls of the [CanIf\_RxIndication()](#_bookmark383).

The CAN hardware receive object is locked until the end of the copy process to the tem- porary or upper layer module buffer. The hardware object will be immediately released after [CanIf\_RxIndication()](#_bookmark383) of [CanIf](#_bookmark8) returns to avoid loss of data.

[CanDrv](#_bookmark5), [CanIf](#_bookmark8) and the upper layer module, which belongs to the received [L-PDU](#_bookmark3), access the same temporary intermediate buffer, which can be located either in the CAN hardware receive object of the [CAN Controller](#_bookmark24) or as temporary buffer in [CanDrv](#_bookmark5).



|  |  |  |
| --- | --- | --- |
| [Yes] | «datastore»  Temporary buffer in CAN Driver | |
|  | |  |
|  | | |



**Figure 7.6: Receive data flow**



Receive Interrupt

Data

normalization necessary?

[No]

Call CanIf\_RxIndication()

Rx L-PDU [Yes]

received in BasicCAN ?

Software filtering

[No]

[Yes]

Data Length Check enabled?

[L-PDU passed]

[No]

Data Length Check failed ?

[No]

[Yes]

Call Det\_ReportRuntimeError() with error code

CANIF\_E\_INVALID\_DATA\_LENGTH

[L-PDU not passed]

Call <User\_RxIndication>() to upper layers

«datastore»

Copy data to L-PDU buffer

<User\_RxIndication>() returns CanIf\_RxIndication() returns

#### Receive indication

A call of [CanIf\_RxIndication()](#_bookmark383) (see [[SWS\_CANIF\_00006](#_bookmark382)]) references in its pa- rameters a newly received CAN L-PDU. If the function [CanIf\_RxIndication()](#_bookmark383) is

called, the CanIf evaluates the CAN L-PDU for acceptance and prepares the [L-SDU](#_bookmark4) for later access by the upper communication layers. The CanIf notifies upper layer modules about this asynchronous event using <User\_RxIndication>() (see [sub-](#_bookmark422) [subsection](#_bookmark422) [8.6.3.3](#_bookmark422) “[<User\_RxIndication>](#_bookmark422)”, [[SWS\_CANIF\_00012](#_bookmark423)]), 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 [CanIfBusMirror-](#_bookmark462) [ingSupport](#_bookmark462)) and has been activated with a call to [CanIf\_EnableBusMirroring](#_bookmark361) [()](#_bookmark361) for a [CAN Controller](#_bookmark24), the [CanIf](#_bookmark8) shall call Mirror\_ReportCanFrame() for each frame reception on that controller that is indicated with [CanIf\_RxIndication](#_bookmark383) [()](#_bookmark383).*](*[*SRS\_Can\_01172*](#_bookmark138)*)*

**[SWS\_CANIF\_00389]** *[*If the function [CanIf\_RxIndication()](#_bookmark383) is called, the [CanIf](#_bookmark8) shall process the Software Filtering on the received L-PDU, if configured (see multi- plicity of [CanIfHrhRangeCfg](#_bookmark549) equals 0*..∗*). If Software Filtering rejects the received L-PDU, the CanIf shall end the receive indication for that call of [CanIf\_RxIndica-](#_bookmark383) [tion()](#_bookmark383).*]()*

Note for [[SWS\_CANIF\_00389](#_bookmark185)]: See [7.20](#_bookmark215).

**[SWS\_CANIF\_00390]** *[*If [CanIf](#_bookmark8) accepts an L-PDU received via [CanIf\_RxIndica-](#_bookmark383) [tion()](#_bookmark383) during Software Filtering (see [[SWS\_CANIF\_00389](#_bookmark185)]), [CanIf](#_bookmark8) shall process the Data Length check afterwards, if configured (see [CanIfPrivateDataLength-](#_bookmark458) [Check](#_bookmark458) and [CanIfRxPduDataLengthCheck](#_bookmark500)).*]()*

For further details, please refer to [section](#_bookmark220) [7.21](#_bookmark220) “[Data Length Check](#_bookmark220)”.

**[SWS\_CANIF\_00297]** *[*If [CanIf](#_bookmark8) has accepted a [L-PDU](#_bookmark3) received via [CanIf\_-](#_bookmark383) [RxIndication()](#_bookmark383) during Data Length Check (see [[SWS\_CANIF\_00390](#_bookmark186)]), [CanIf](#_bookmark8) shall copy the number of bytes according to the configured Data Length (see [ECUC\_CanIf\_00599](#_bookmark499)) to the static receive buffer, if configured for that L-PDU (see [[SWS\_CANIF\_00198](#_bookmark190)], [ECUC\_CanIf\_00600](#_bookmark502)).*]()*

**[SWS\_CANIF\_00851]** *[*If MetaData is configured for a received [L-SDU](#_bookmark4), [CanIf](#_bookmark8) shall copy the PDU payload to the static receive buffer and the CAN ID to the Meta- DataItem of type CAN\_ID\_32.*]()*

**[SWS\_CANIF\_00056]** *[*If [CanIf](#_bookmark8) accepts a [L-PDU](#_bookmark3) received via [CanIf\_-](#_bookmark383) [RxIndication()](#_bookmark383) during Data Length Check (see [[SWS\_CANIF\_00390](#_bookmark186)], [[SWS\_CANIF\_00026](#_bookmark221)]), [CanIf](#_bookmark8) shall identify if a target upper layer module was configured (see configuration descrption of [[SWS\_CANIF\_00012](#_bookmark423)], [CanIfRxPdu-](#_bookmark505) [UserRxIndicationUL](#_bookmark505), [CanIfRxPduUserRxIndicationName](#_bookmark504)) to be called with its providing receive indication service for the received [L-SDU](#_bookmark4).*]()*

**[SWS\_CANIF\_00135]** *[*If a target upper layer module was configured to be called with its providing receive indication service (see [[SWS\_CANIF\_00056](#_bookmark187)]), the CanIf shall call this configured receive indication callback service (see [CanIfRxPduUserRxIndi-](#_bookmark504) [cationName](#_bookmark504)) and shall provide the parameters required for upper layer notification callback functions (see [[SWS\_CANIF\_00012](#_bookmark423)]) based on the parameters of [CanIf\_-](#_bookmark383) [RxIndication()](#_bookmark383).*](*[*SRS\_BSW\_00325*](#_bookmark83)*)*

Note: A single receive L-PDU can only be assigned to a single receive indication call- back service (refer to multiplicity of [CanIfRxPduUserRxIndicationName](#_bookmark504)).

Overview: CanIf performs the following steps at a call of [CanIf\_RxIndication()](#_bookmark383):

* Software Filtering (only BasicCAN), if configured
* Data Length Check, if configured
* buffer received [L-SDU](#_bookmark4) if configured
* call upper layer receive indication callback service, if configured.

#### Read received data

The read received data API [CanIf\_ReadRxPduData()](#_bookmark289) (see [[SWS\_CANIF\_00194](#_bookmark288)]) is a common interface for upper layer modules to read [CAN L-SDUs](#_bookmark4) recently received from the CAN network. The upper layer modules initiate the receive request only via [CanIf](#_bookmark8) services without direct access to [CanDrv](#_bookmark5). The initiated receive request is suc- cessfully completed, if [CanIf](#_bookmark8) wrote the received [L-SDU](#_bookmark4) into the upper layer module I-PDU buffer.

The function [CanIf\_ReadRxPduData()](#_bookmark289) makes reading out data without dependence of reception event (RxIndication) possible. When it is enabled at configuration time (see [CanIfPublicReadRxPduDataApi](#_bookmark467)), not necessarily a receive indication service for the same [L-SDU](#_bookmark4) has to be configured (see [CanIfRxPduUserRxIndicationUL](#_bookmark505)). If needed, the receive indication can be enabled, too.

By this way the type of mechanism to receive [L-SDUs](#_bookmark4) (in the upper layer modules of [CanIf](#_bookmark8)) can be chosen at configuration time by the parameter [CanIfRxPduUser-](#_bookmark505) [RxIndicationUL](#_bookmark505) and parameter [CanIfRxPduReadData](#_bookmark502) according to the needs of the upper layer module, to which the corresponding receive [L-SDU](#_bookmark4) belongs to. For details please refer to [section](#_bookmark447) [9.9](#_bookmark447) “[Read received data](#_bookmark447)”.

**[SWS\_CANIF\_00198]** *[*If the configuration parameter [CanIfPublicReadRxPdu-](#_bookmark467) [DataApi](#_bookmark467) is set to TRUE, [CanIf](#_bookmark8) shall store each received [L-SDU](#_bookmark4), at which [CanI-](#_bookmark502) [fRxPduReadData](#_bookmark502) is enabled, into a receive [L-SDU](#_bookmark4) buffer. This means that if the con- figuration parameter [CanIfRxPduReadData](#_bookmark502) is set to TRUE, [CanIf](#_bookmark8) has to allocate a receive [L-SDU](#_bookmark4) buffer for this receive [L-SDU](#_bookmark4).*]()*

**[SWS\_CANIF\_00199]** *[*After call of [CanIf\_RxIndication()](#_bookmark383) and passing of soft- ware filtering and Data Length Check, [CanIf](#_bookmark8) shall store the received [L-SDU](#_bookmark4) in this receive [L-SDU](#_bookmark4) buffer. During the call of [CanIf\_ReadRxPduData()](#_bookmark289) the assigned receive [L-SDU](#_bookmark4) buffer containing a recently received [L-SDU](#_bookmark4), [CanIf](#_bookmark8) shall avoid pre- emptive receive [L-SDU](#_bookmark4) buffer access events (refer to [[SWS\_CANIF\_00064](#_bookmark193)]) to that receive [L-SDU](#_bookmark4) buffer.*]()*

#### Read Tx/Rx notification status

In addition to the notification callback functions [CanIf](#_bookmark8) provides the API service [CanIf\_ReadTxNotifStatus()](#_bookmark294) (see [[SWS\_CANIF\_00202](#_bookmark293)]) to read the transmit confirmation status of any transmit [L-SDU](#_bookmark4) and the API service [CanIf\_ReadRxNo-](#_bookmark299) [tifStatus()](#_bookmark299) is provided to read the receive indication status of any receive [L-SDU](#_bookmark4).

[CanIf](#_bookmark8)’s API services [CanIf\_ReadTxNotifStatus()](#_bookmark294) (see [[SWS\_CANIF\_00202](#_bookmark293)]) and [CanIf\_ReadRxNotifStatus()](#_bookmark299) (see [[SWS\_CANIF\_00230](#_bookmark298)]) can be en- abled/disabled globally or per [L-SDU](#_bookmark4) at pre-compile time configuration using the con- figuration parameters [CanIfPublicReadTxPduNotifyStatusApi](#_bookmark469), [CanIfPubli-](#_bookmark468) [cReadRxPduNotifyStatusApi](#_bookmark468), [CanIfTxPduReadNotifyStatus](#_bookmark487), and [CanI-](#_bookmark503) [fRxPduReadNotifyStatus](#_bookmark503).

**[SWS\_CANIF\_00472]** *[*If configuration parameter [CanIfPublicReadTxPduNoti-](#_bookmark469) [fyStatusApi](#_bookmark469) is set to TRUE, [CanIf](#_bookmark8) shall store the current notification status for each transmit [L-SDU](#_bookmark4).*]()*

**[SWS\_CANIF\_00473]** *[*If configuration parameter [CanIfPublicReadRxPduNoti-](#_bookmark468) [fyStatusApi](#_bookmark468) is set to TRUE, [CanIf](#_bookmark8) shall store the current notification status for each receive [L-SDU](#_bookmark4).*]()*

Rationale for [[SWS\_CANIF\_00391](#_bookmark378)] and [[SWS\_CANIF\_00393](#_bookmark295)] respectively [[SWS\_CANIF\_00392](#_bookmark384)] and [[SWS\_CANIF\_00394](#_bookmark300)]: This ’read-and-consume’ be- havior ensures, that at least one successful transmit or receive event occurred after last call of this service.

#### Data integrity

**[SWS\_CANIF\_00064] Shared code shall be reentrant** *[*[CanIf](#_bookmark8) shall protect preemp- tive events, which access shared resources, that could be changed during [CanIf](#_bookmark8)’s event handling, against each other.*]([SRS\_BSW\_00312](#_bookmark81))*

Rationale: An attempt to update the data in the upper layer module buffers as well as in [CanIf](#_bookmark8)’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 net- work systems interrupts, which have the needs to transmit and receive [L-PDUs](#_bookmark3) on the network.

Handling of shared transmit and receive [L-PDU](#_bookmark3)/[L-SDU](#_bookmark4) buffers are critical issues for the implementation of [CanIf](#_bookmark8). Therefore [CanIf](#_bookmark8) shall ensure data integrity and thus use appropriate mechanisms for access to shared resources like transmission/reception [L-PDU](#_bookmark3)/[L-SDU](#_bookmark4) buffers. Preemptive events, i.e. transmission and reception event from other [CAN Controllers](#_bookmark24) could compromise data integrity by writing into the same [L-PDU](#_bookmark3)/[L-SDU](#_bookmark4) buffer.

[CanIf](#_bookmark8) can e.g. use [CanDrv](#_bookmark5) services to enable (Can\_EnableControllerInter- rupts()) and disable (Can\_Disable-ControllerInterrupts()) CAN interrupts and its notifications at entry and exit of the critical sections separately for each [CAN](#_bookmark24) [Controller](#_bookmark24). If there are common resources for multiple [CAN Controllers](#_bookmark24), 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()](#_bookmark279) must be able to operate re-entrant to allow multiple transmit request calls caused by different preemptive events of different [L-PDUs](#_bookmark3)/[L-SDUs](#_bookmark4). [CanDrv](#_bookmark5)’s transmit request API Can\_Write() operates re-entrant as well.

#### CAN Controller Mode

##### General Functionality

[CanIf](#_bookmark8) provides services for controlling the communication mode of all supported [CAN Controllers](#_bookmark24) represented by the underlying [CanDrv](#_bookmark5). This means that all [CAN](#_bookmark24) [Controllers](#_bookmark24) are controlled by the corresponding provided API services to request and read the current controller mode.

The [CAN Controller](#_bookmark24) status may be changed at request of the upper layer by the calling of [CanIf\_SetControllerMode()](#_bookmark264) service. The request is passed by [CanIf](#_bookmark8) via the [CanDrv](#_bookmark5) API to the addressed [CAN Controller](#_bookmark24).

The consistent management of all [CAN Controllers](#_bookmark24) connected at one CAN network is the task of [CanSm](#_bookmark10). By this way [CanSm](#_bookmark10) is responsible to set all [CAN Controllers](#_bookmark24) of one CAN network sequentially to sleep mode or to wake them up.

[CanIf](#_bookmark8) accepts every state transition request by calling the function [CanIf\_SetCon-](#_bookmark264) [trollerMode()](#_bookmark264) or [CanIf\_ControllerBusOff()](#_bookmark389). [CanIf](#_bookmark8) does not decide if a re- quested mode transition of the [CAN Controller](#_bookmark24) is valid or not. [CanIf](#_bookmark8) only interacts with [CanDrv](#_bookmark5) by fetching the current mode and execution of requested mode transi- tions.

This network related state machine is implemented in [CanSm](#_bookmark10). Refer to [[3](#_bookmark38)]. [CanIf](#_bookmark8) only stores the requested mode and executes the requested transition.

Hint: As optimisation to avoid frequent requests to [CanDrv](#_bookmark5) for internal use the last state indicated by [CanIf\_ControllerModeIndication()](#_bookmark402) and Can\_GetCon- trollerMode() could be stored per controller.

Hint: It has to be regarded that not only [CanSm](#_bookmark10) is able to request CAN Controller Mode changes.

##### CAN Controller Operation Modes

According to the requested operation mode by [CanSm](#_bookmark10), [CanIf](#_bookmark8) forwards request [Can-](#_bookmark5) [Drvs](#_bookmark5).

**[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()](#_bookmark279) is assigned to that [CAN Controller](#_bookmark24), then the call of [CanIf\_Transmit()](#_bookmark279) does not result in a call of Can\_Write() (see [[SWS\_CANIF\_00317](#_bookmark280)]) and returns E\_NOT\_OK.*] ()*

**[SWS\_CANIF\_00485]** *[*If a controller mode referenced by ControllerId enters state CAN\_CS\_STOPPED, then [CanIf](#_bookmark8) shall clear the CanIf transmit buffers assigned to the [CAN Controller](#_bookmark24) corresponding.*]()*

**[SWS\_CANIF\_00739]** *[*If a controller mode referenced by ControllerId enters state CAN\_CS\_STOPPED, then [CanIf](#_bookmark8) 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 [CanIfPublicTx-](#_bookmark472) [ConfirmPollingSupport](#_bookmark472) is enabled, [CanIf](#_bookmark8) shall also clear the information about a TxConfirmation (see [[SWS\_CANIF\_00740](#_bookmark181)]).*]()*

Note: This ensures, that for each PDU, which shall be transmitted via [CanIf\_Trans-](#_bookmark279) [mit()](#_bookmark279), either a positive or negative <User\_TxConfirmation>() is called.

**[SWS\_CANIF\_00724]** *[*When callback [CanIf\_ControllerBusOff(Control-](#_bookmark389) [lerId)](#_bookmark389) is called, the [CanIf](#_bookmark8) shall call CanSM\_ControllerBusOff(Control- lerId) of the [CanSm](#_bookmark10) or a *CDD* (see [[SWS\_CANIF\_00559](#_bookmark427)], [[SWS\_CANIF\_00560](#_bookmark428)]).*] ()*

Note for [[SWS\_CANIF\_00724](#_bookmark197)]: See [subsubsection](#_bookmark432) [8.6.3.9](#_bookmark432) “[<User\_ControllerMod-](#_bookmark432) [eIndication>](#_bookmark432)”.

**[SWS\_CANIF\_00711]** *[*When callback [CanIf\_ControllerModeIndication](#_bookmark402) [(ControllerId, ControllerMode)](#_bookmark402) is called, [CanIf](#_bookmark8) shall call CanSm\_Con- trollerModeIndication(ControllerId, ControllerMode) of the [CanSm](#_bookmark10) or a *CDD* (see [[SWS\_CANIF\_00691](#_bookmark433)], [[SWS\_CANIF\_00692](#_bookmark434)]).*]()*

Note for [[SWS\_CANIF\_00711](#_bookmark198)]: See [subsubsection](#_bookmark432) [8.6.3.9](#_bookmark432) “[<User\_ControllerMod-](#_bookmark432) [eIndication>](#_bookmark432)”.

**[SWS\_CANIF\_00712]** *[*When callback [CanIf\_TrcvModeIndication](#_bookmark404) [(Transceiver, TransceiverMode)](#_bookmark404) is called, [CanIf](#_bookmark8) shall call CanSM\_- TransceiverModeIndication(TransceiverId, TransceiverMode) of the [CanSm](#_bookmark10) or a *CDD* (see [[SWS\_CANIF\_00697](#_bookmark436)], [[SWS\_CANIF\_00698](#_bookmark437)]).*]()*

Note for [[SWS\_CANIF\_00712](#_bookmark199)]: See [subsubsection](#_bookmark432) [8.6.3.9](#_bookmark432) “[<User\_ControllerMod-](#_bookmark432) [eIndication>](#_bookmark432)”.

##### Controller Mode Transitions

The API for state change requests to the [CAN Controller](#_bookmark24) behaves in an asyn- chronous 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](#_bookmark5) calls function [CanIf\_ControllerModeIndication()](#_bookmark402) and [CanIf](#_bookmark8) in turn calls function

<User\_ControllerModeIndication>(). If CAN transitions very fast, [CanIf\_-](#_bookmark402) [ControllerModeIndication()](#_bookmark402) can be called during [CanIf\_SetController-](#_bookmark264) [Mode()](#_bookmark264). This is implementation specific.

Unsuccessful or no mode transitions of the [CAN Controllers](#_bookmark24) 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](#_bookmark8) can poll the current Controller Mode by [CanIf\_Get-](#_bookmark269) [ControllerMode()](#_bookmark269).

Not all types of [CAN Controllers](#_bookmark24) support *Sleep* and *Wake-Up Mode*. These modes are then encapsulated by [CanDrv](#_bookmark5) by providing hardware independent oper- ation modes via its interface, which has to be managed by [CanIf](#_bookmark8).

Note: It is possible that during transition from CAN\_CS\_STOPPED to CAN\_CS\_SLEEP [CAN Controller](#_bookmark24) may indicate a wake-up interrupt to the ECU Integration Code.

[CanIf](#_bookmark8) distinguishes between internal initiated CAN controller wake-up request (inter- nal request) and network wake-up request (external request). The internal request is initiated by call of [CanIf](#_bookmark8)’s function [CanIf\_SetControllerMode(ControllerId,](#_bookmark264) [CAN\_CS\_STARTED)](#_bookmark264) and it is an internal asynchronous request. The external request is a CAN controller event, which is notified by [CanDrv](#_bookmark5) or [CanTrcv](#_bookmark12) to the ECU Inte- gration Code. For details see respective UML diagram in the chapter "CAN Wakeup Sequences" of document [[13](#_bookmark48)].

##### Wake-up

The ECU supports wake-up over CAN network, regardless of the used wake-up method (directly about [CAN Controller](#_bookmark24) or [CAN Transceiver](#_bookmark12)), only if the [CAN](#_bookmark24) [Controller](#_bookmark24) and [CAN Transceiver](#_bookmark12) 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](#_bookmark24) is stopped. Thus, the wake-up interrupt can be enabled.

###### Wake-up detection

If *wake-up support* is enabled (see [ECUC\_CanIf\_00843](#_bookmark477)) [CanIf](#_bookmark8) is notified by the Inte- gration Code about a detected CAN wake-up by the service [CanIf\_CheckWakeup()](#_bookmark338) (see CAN Wakeup Sequences of [[13](#_bookmark48)]).

In case of a CAN bus "*wake-up*" event the function [CanIf\_CheckWakeup(Wakeup-](#_bookmark338) [Source)](#_bookmark338) may be called during execution of EcuM\_CheckWakeup(WakeupSource) (see wake-up sequence diagrams of [EcuM](#_bookmark16)). [CanIf](#_bookmark8) in turn checks by configured input reference to EcuMWakeupSource in [CanDrvs](#_bookmark5), which [CanDrvs](#_bookmark5) have to be checked. [CanIf](#_bookmark8) gets this information via reference [CanIfCtrlCanCtrlRef](#_bookmark528).

The Communication Service, which is called, belongs to the service defined during configuration (see [CanIfDispatchCfg](#_bookmark509)). In this way [EcuM](#_bookmark16) as well as [CanSm](#_bookmark10) 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](#_bookmark338) [WakeupSource)](#_bookmark338) is invoked, [CanIf](#_bookmark8) shall query [CanDrvs](#_bookmark5) / [CanTrcvs](#_bookmark12) 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](#_bookmark8) 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](#_bookmark8), then [CanIf\_CheckWakeup()](#_bookmark338) shall return E\_OK.*]()*

**[SWS\_CANIF\_00678]** *[*If all calls of Can\_CheckWakeup() or CanTrcv\_Check- Wakeup() return E\_NOT\_OK to [CanIf](#_bookmark8), then [CanIf\_CheckWakeup()](#_bookmark338) shall return E\_NOT\_OK.*]()*

###### Wake-up Validation

Note: When a [CAN Controller](#_bookmark24) / [CAN Transceiver](#_bookmark12) 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](#_bookmark16) (or a *CDD*) switches on the corresponding [CAN Con-](#_bookmark24) [troller](#_bookmark24) ([CanIf\_SetControllerMode()](#_bookmark264)) and [CAN Transceiver](#_bookmark12) ([CanIf\_Set-](#_bookmark322) [TrcvMode()](#_bookmark322)) (For more details see chapter 9 of [[13](#_bookmark48)]).

Attention: [CanIf](#_bookmark8) 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](#_bookmark36)]) a direct transition from mode CAN\_CS\_SLEEP to CAN\_CS\_STARTED is not allowed.

**SWS\_CANIF\_00286]** *[*If [CanIfPublicWakeupCheckValidSupport](#_bookmark474) equals TRUE, [CanIf](#_bookmark8) enables the detection for CAN wake-up validation. Therefore, [CanIf](#_bookmark8) stores the event of the first valid call of [CanIf\_RxIndication()](#_bookmark383) of a [CAN Controller](#_bookmark24) which has been set to CAN\_CS\_STARTED. The first call of [CanIf\_RxIndication()](#_bookmark383) is valid:

* + - * + only for received NM messages if [CanIfPublicWakeupCheckValidByNM](#_bookmark473) is

TRUE

* + - * + for all received messages corresponding to a configured Rx PDU if [CanIfPub-](#_bookmark473) [licWakeupCheckValidByNM](#_bookmark473) is FALSE.

*]([SRS\_Can\_01151](#_bookmark134))*

**[SWS\_CANIF\_00179]** *[*<User\_ValidateWakeupEvent>(sources) shall be called during [CanIf\_CheckValidation(WakeupSource)](#_bookmark341), whereas sources is set to WakeupSource, if the event of the first called [CanIf\_RxIndication()](#_bookmark383) is stored in [CanIf](#_bookmark8) at the corresponding [CAN Controller](#_bookmark24).*](*[*SRS\_Can\_01136*](#_bookmark130)*)*

Note: The parameter of the function <User\_ValidateWakeupEvent>() is of type:

* + - * + sources: EcuM\_WakeupSourceType (see [[13](#_bookmark48)])

**[SWS\_CANIF\_00756]** *[*When controller mode is set to CAN\_CS\_SLEEP the stored event from previous wake-up (first call of [CanIf\_RxIndication](#_bookmark383)) shall be cleared (see [[SWS\_CANIF\_00179](#_bookmark205)]).*]()*

#### PDU channel mode control

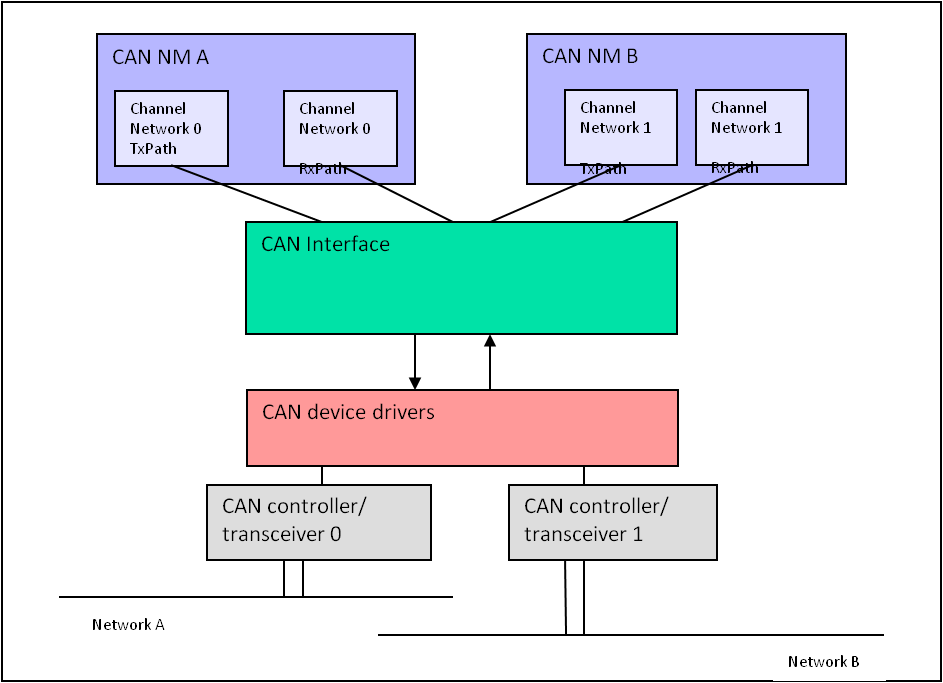
##### PDU channel groups

Each [L-PDU](#_bookmark3) is assigned to one dedicated physical CAN channel connected to one [CAN Controller](#_bookmark24) and one CAN network. By this way all [L-PDUs](#_bookmark3) belonging to one [Physical Channel](#_bookmark32) can be controlled on the view of handling logically single [L-PDU](#_bookmark3) channel groups. Those logical groups represent all [L-PDUs](#_bookmark3) of one ECU connected to one underlying CAN network.

[Figure](#_bookmark208) [7.7](#_bookmark208) below shows one possible usage of [L-PDU](#_bookmark3) channel group and its relation to the upper layers and/or networks.

An [L-PDU](#_bookmark3) can only be assigned to one channel group.

Typical users like [PduR](#_bookmark22) or the Network Management are responsible for controlling the PDU operation modes.



**Figure 7.7:** **Channel PDU groups**

##### PDU channel modes

[CanIf](#_bookmark8) provides the services [CanIf\_SetPduMode()](#_bookmark304) and [CanIf\_GetPduMode()](#_bookmark309) to prevent the processing of

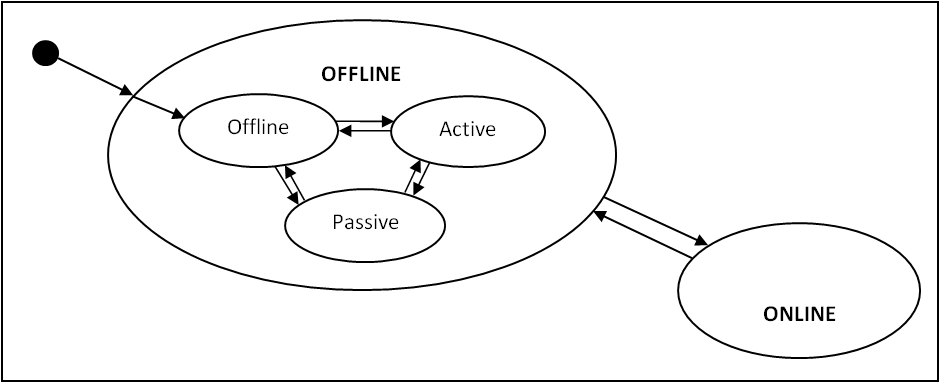
* all [Transmit L-PDUs](#_bookmark27) belonging to one logical channel,
* all [Transmit L-PDUs](#_bookmark27) and [Receive L-PDUs](#_bookmark26) belonging to one logical channel.

Changing the PDU channel mode is only allowed in case corresponding controller mode equals CAN\_CS\_STARTED (refer to [[SWS\_CANIF\_00874](#_bookmark307)]).

While CANIF\_ONLINE and CANIF\_OFFLINE affecting the whole communicatoin the PDU channel modes CANIF\_TX\_OFFLINE and CANIF\_TX\_OFFLINE\_ACTIVE en- able/disable transmission path seperately.

[CanIf](#_bookmark8) provides information about the current PDU channel mode via the service

[CanIf\_GetPduMode()](#_bookmark309).



**Figure 7.8:** **PDU channel mode control**

[Figure](#_bookmark210) [7.8](#_bookmark210) shows a diagram with possible PDU channel modes. Each [L-PDU](#_bookmark3) chan- nel can be in CANIF\_OFFLINE (no communication), CANIF\_TX\_OFFLINE (passive mode => listen without sending), CANIF\_TX\_OFFLINE\_ACTIVE (simulated transmis- sion with listening), and CANIF\_ONLINE (full communication). The default state is the CANIF\_OFFLINE mode.

###### CANIF\_OFFLINE

**[SWS\_CANIF\_00864]** *[*During initialization [CanIf](#_bookmark8) shall switch every channel to

CANIF\_OFFLINE.*]()*

**[SWS\_CANIF\_00865]** *[*If [CanIf\_SetControllerMode(ControllerId, CAN\_-](#_bookmark264) [CS\_SLEEP)](#_bookmark264) is called, [CanIf](#_bookmark8) shall set the PDU channel mode of the corresponding channel to CANIF\_OFFLINE.*]()*

**[SWS\_CANIF\_00073]** *[*For [Physical Channels](#_bookmark32) switching to CANIF\_OFFLINE

mode [CanIf](#_bookmark8) shall:

* + - * + prevent forwarding of transmit requests [CanIf\_Transmit()](#_bookmark279) of associated [L-](#_bookmark3)

[-PDUs](#_bookmark3) to [CanDrv](#_bookmark5) (return E\_NOT\_OK to the calling upper layer modules),

* + - * + clear the corresponding [CanIf](#_bookmark8) transmit buffers,
        + prevent invocation of receive indication callback services of the upper layer mod- ules,
        + prevent invocation of transmit confirmation callback services of the upper layer modules.

*]()*

**[SWS\_CANIF\_00866]** *[*If [CanIf\_SetControllerMode(ControllerId, CAN\_-](#_bookmark264) [CS\_STOPPED)](#_bookmark264) or [CanIf\_ControllerBusOff(ControllerId)](#_bookmark389) is called, [CanIf](#_bookmark8) shall set the PDU channel mode of the corresponding channel to CANIF\_TX\_OF- FLINE.*]()*

**[SWS\_CANIF\_00489]** *[*For [Physical Channels](#_bookmark32) switching to CANIF\_TX\_OFFLINE

mode [CanIf](#_bookmark8) shall:

* + - * + prevent forwarding of transmit requests [CanIf\_Transmit()](#_bookmark279) of associated [L-](#_bookmark3)

[-PDUs](#_bookmark3) to [CanDrv](#_bookmark5) (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 mod- ules.

*]()*

The *BusOff* notification is implicitly suppressed in case of CANIF\_OFFLINE and CANIF\_TX\_OFFLINE due to the fact, that no [L-PDUs](#_bookmark3) can be transmitted and thus the [CAN Controller](#_bookmark24) is not able to go in *BusOff* mode by newly requested [L-PDUs](#_bookmark3) for transmission.

**[SWS\_CANIF\_00118]** *[*If those [Transmit L-PDUs](#_bookmark27), which are already waiting for transmission in the [CAN Transmit Hardware Object](#_bookmark29), will be transmitted imme- diately after change to CANIF\_TX\_OFFLINE or CANIF\_OFFLINE mode and a subse- quent *BusOff* event occurs, [CanIf](#_bookmark8) does not prohibit execution of the *BusOff* notifica- tion <User\_ControllerBusOff>(ControllerId).*]()*

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

###### CANIF\_ONLINE

**[SWS\_CANIF\_00075]** *[*For [Physical Channels](#_bookmark32) switching to CANIF\_ONLINE mode [CanIf](#_bookmark8) shall:

* + - * + enable forwarding of transmit requests [CanIf\_Transmit()](#_bookmark279) of associated [L-](#_bookmark3)

[-PDUs](#_bookmark3) to [CanDrv](#_bookmark5),

* + - * + enable invocation of receive indication callback services of the upper layer mod- ules,
        + enable invocation of transmit confirmation callback services of the upper layer modules.

*]()*

###### CANIF\_OFFLINE\_ACTIVE

If CanIfTxOfflineActiveSupport = TRUE [CanIf](#_bookmark8) provides simulation of suc- cessful transmission by CANIF\_TX\_OFFLINE\_ACTIVE mode. This mode is enabled by call of [CanIf\_SetPduMode(ControllerId, CANIF\_TX\_OFFLINE\_ACTIVE)](#_bookmark304) and only affects the transmission path.

**[SWS\_CANIF\_00072]** *[*For every [L-PDU](#_bookmark3) assigned to a channel which is in CANIF\_- TX\_OFFLINE\_ACTIVE mode [CanIf](#_bookmark8) shall call the transmit confirmation callback ser- vices of the upper layer modules immediately instead of buffering or forwarding of the [L-PDUs](#_bookmark3) to [CanDrv](#_bookmark5) during the call of [CanIf\_Transmit()](#_bookmark279).*]()*

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](#_bookmark214)]).

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.

#### Software receive filter

Not all [L-PDUs](#_bookmark3), which may pass the hardware acceptance filter and therefore are successful received in *BasicCAN* [Hardware Objects](#_bookmark29), are defined as [Receive L-](#_bookmark26)

[-PDUs](#_bookmark26) and thus needed from the corresponding ECU. [CanIf](#_bookmark8) optionally filters out these [L-PDUs](#_bookmark3) 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](#_bookmark3) from the [HRH](#_bookmark18) and [CanId](#_bookmark7) currently being processed. After the [L-PDU](#_bookmark3) is found, [CanIf](#_bookmark8) accepts the reception and enables upper layers to access [L-SDU](#_bookmark4) information directly.

##### Software filtering concept

The configuration tool handles the information about hardware acceptance filter set- tings. The most important settings are the number of the L-PDU hardware objects and their range. The outlet range defines, which [Receive L-PDUs](#_bookmark26) belongs to each [Hardware Receive Object](#_bookmark29). The following definitions are possible:

* a single [Receive L-PDU](#_bookmark26) (*FullCAN* reception),
* a list of [Receive L-PDUs](#_bookmark26) or
* one or multiple ranges of [Receive L-PDUs](#_bookmark26) can be linked to a [Hardware Re-](#_bookmark29) [ceive Object](#_bookmark29) (*BasicCAN* reception).

For definition of range reception it is necessary to define at least one [Rx L-PDU](#_bookmark26) where the [CanId](#_bookmark7) or the complete ID range is inside the defined range.

**[SWS\_CANIF\_00645]** *[*A range of [CanIds](#_bookmark7) which shall pass the software receive filter shall either be defined by its upper limit (see [CanIfHrhRangeRxPduUpperCanId](#_bookmark554)) and lower limit (see [CanIfHrhRangeRxPduLowerCanId](#_bookmark552)) [CanId](#_bookmark7), or by a base ID (see [CanIfHrhRangeBaseId](#_bookmark550)) and a mask that defines the relevant bits of the base ID (see [CanIfHrhRangeMask](#_bookmark551)).*]()*

Note: Software receive filtering is optional (see multiplicity of 0*..∗* in [Can-](#_bookmark549) [IfHrhRangeCfg](#_bookmark549)).

**[SWS\_CANIF\_00646]** *[*Each configurable range of [CanIds](#_bookmark7) (see [[SWS\_CANIF\_00645](#_bookmark217)]), which shall pass the software receive filter, shall be configurable either for *Standard CAN IDs* or *Extended CAN IDs* via [Can-](#_bookmark553) [IfHrhRangeRxPduRangeCanIdType](#_bookmark553).*]()*

[Receive L-PDUs](#_bookmark26) 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](#_bookmark7) for every [Hardware](#_bookmark29) [Receive Object](#_bookmark29) ([HRH](#_bookmark18)). The corresponding list can be derived by the [HRH](#_bookmark18), if multiple *BasicCAN* objects are used. The subsequent filtering is the search through one list of multiple [CanIds](#_bookmark7) by comparing them with the new received [CanId](#_bookmark7). In case of a hit the [Receive L-PDU](#_bookmark26) is derived from the found [CanId](#_bookmark7).

**[SWS\_CANIF\_00030]** *[*If the [CanId](#_bookmark7) of the received [L-PDU](#_bookmark3) in the [HRH](#_bookmark18) is configured to be received, then [CanIf](#_bookmark8) shall accept this [L-PDU](#_bookmark3) and the software filtering algorithm shall derive the corresponding [Receive L-PDU](#_bookmark26) from the found [CanId](#_bookmark7).*]([SRS\_Can\_-](#_bookmark117)* [*01018*](#_bookmark117)*)*



**Figure 7.9: Software filtering example**

**[SWS\_CANIF\_00852]** *[*If a range is (partly) contained in another range, or a single [CanId](#_bookmark7) is contained in a range, the software filter shall select the [L-PDU](#_bookmark3) based on the following assumptions:

* A single [CanId](#_bookmark7) is always more relevant than a range.
* A smaller range is more relevant than a larger range.

*]()*

##### Software filter algorithms

The choice of suitable software search algorithms it is up to the implementation of [CanIf](#_bookmark8). According to the wide range of possible receive *BasicCAN* operations provided by the [CAN Controller](#_bookmark24) 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.

#### 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](#_bookmark8) shall accept all received L-PDUs (see [[SWS\_CANIF\_00390](#_bookmark186)]) with a Data Length value equal or greater then the configured Data Length value (see [CanIfRxPduDataLength](#_bookmark499)).*](*[*SRS\_Can\_01005*](#_bookmark112)*)*

**[SWS\_CANIF\_00902]** *[*The Data Length Check shall be processed if it is enabled globally (see [CanIfPrivateDataLengthCheck](#_bookmark458)) and not disabled individually per PDU (see [CanIfRxPduDataLengthCheck](#_bookmark500)).*]()*

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](#_bookmark221)]), [CanIf](#_bookmark8) shall report runtime error code CANIF\_E\_IN- VALID\_DATA\_LENGTH to the Det\_ReportRuntimeError() service of the DET module.*]()*

**[SWS\_CANIF\_00829]** *[*[CanIf](#_bookmark8) shall pass the received (see [[SWS\_CANIF\_00006](#_bookmark382)]) length value to the target upper layer module (see [[SWS\_CANIF\_00135](#_bookmark188)]), if the Data Length Check is passed.*]()*

**[SWS\_CANIF\_00830]** *[*[CanIf](#_bookmark8) shall pass the received (see [[SWS\_CANIF\_00006](#_bookmark382)]) length value to the target upper layer module (see [[SWS\_CANIF\_00135](#_bookmark188)]), if the Data Length Check is not configured (see [CanIfPrivateDataLengthCheck](#_bookmark458) and [CanI-](#_bookmark500) [fRxPduDataLengthCheck](#_bookmark500))*]()*

#### L-SDU dispatcher to upper layers

Rationale: At transmission side the [L-SDU](#_bookmark4) dispatcher has to find out the correspond- ing Tx confirmation callback service of the target upper layer module. At reception side each [L-SDU](#_bookmark4) belongs to one single upper layer module as destination. This relation is

assigned statically at configuration time. The task of the [L-SDU](#_bookmark4) dispatcher inside of [CanIf](#_bookmark8) is to find out the customer for a received [L-SDU](#_bookmark4) and to dispatch the indica- tions 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.

#### 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 forward- ing of detected events by the CanDrv are the same like for interrupt operation (see [section](#_bookmark372) [8.4](#_bookmark372) “[Callback notifications](#_bookmark372)”).

The user has to consider, that the CanIf has to be able to perform notification ser- vices 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, subse- quent transmit buffering takes place (refer [section](#_bookmark161) [7.11](#_bookmark161) “[Transmit buffering](#_bookmark161)”).

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

#### Multiple CAN Driver support

[CanIf](#_bookmark8) needs a specific mapping to cover multiple [CanDrv](#_bookmark5) to provide a common inter- face to upper layers. Thus, [CanIf](#_bookmark8) must dispatch all actions up-down to the APIs of the corresponding [CanDrv](#_bookmark5) and underlying [CAN Controller](#_bookmark24)(s). For the way down-up [CanIf](#_bookmark8) has to provide adequate callback notifications to differentiate between multiple [CanDrvs](#_bookmark5).

Each [CanDrv](#_bookmark5) supports a certain number of underlying [CAN Controllers](#_bookmark24) and a fixed number of [HTHs](#_bookmark19)/[HRHs](#_bookmark18). Each [CanDrv](#_bookmark5) has an own numbering area, which starts always at zero for [CAN Controllers](#_bookmark24) and [HTHs](#_bookmark19). [CanIf](#_bookmark8) has to derive the corresponding [CanDrv](#_bookmark5) from the [L-SDU](#_bookmark4) 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](#_bookmark5) can be enabled and disabled by the configuration parameter [CanIfPublicMultipleDrvSupport](#_bookmark465).

##### Transmit requests by using multiple CAN Drivers

Each [Transmit L-PDU](#_bookmark27) enables [CanIf](#_bookmark8) to derive the corresponding [CAN Con-](#_bookmark24) [troller](#_bookmark24) and implicitly [CanDrv](#_bookmark5) serving the affected [Hardware Unit](#_bookmark25). Resolving of these dependencies is possible because of the construction of the *CAN Controller Handle*: it combines *CanDrv Handle* and the corresponding [CAN Controller](#_bookmark24) in the [Hardware Unit](#_bookmark25).

At configuration time a CAN Controller Handle will be mapped to each [CAN Con-](#_bookmark24) [troller](#_bookmark24). The sequence diagram [Figure](#_bookmark227) [7.10](#_bookmark227) below demonstrates two transmit re- quests directed to different [CanDrvs](#_bookmark5). [CanIf](#_bookmark8) needs only to select the corresponding [CanDrv](#_bookmark5) in order to call the correct API service.

Note: [Figure](#_bookmark227) [7.10](#_bookmark227) and the following table serve only as an example. Finally, it is up to the implementation to access the correct APIs of underlying [CanDrvs](#_bookmark5).

CanIf User

«mod...

CanIf

Can\_99\_Ext1: Can

«Peripheral»

CanController A: CanController

Can\_99\_Ext2: Can

«Peripheral»

CanController B: CanController

alt CAN Controller A/B [CAN Controller A used]

CanIf\_Transmit(Std\_ReturnType, PduIdType, const PduInfoType\*)

Can\_Write(Std\_ReturnType, Can\_HwHandleType, const Can\_PduType\*)

Copy L-PDU in CAN Hardware A()

Copy L-PDU in CAN Hardware A()

Can\_Write()

CanIf\_Transmit()

[CAN Controller B used]

CanIf\_Transmit(Std\_ReturnType, PduIdType, const PduInfoType\*)

Can\_Write(Std\_ReturnType, Can\_HwHandleType, const Can\_PduType\*)

Copy L-PDU in CAN Hardware B()

Copy L-PDU in CAN Hardware B()

Can\_Write()

CanIf\_Transmit()

**Figure 7.10:** **Transmission request with multiple CAN Drivers - simplified**

|  |  |
| --- | --- |
| **Operations called** | **Description** |
| [CanIf\_Transmit](#_bookmark279) [(PduId\_1,](#_bookmark279) [PduInfoPtr\_1)](#_bookmark279) | Upper layer initiates a *transmit request*. The PduId is used for  tracing the requested [CAN Controller](#_bookmark24) and then to serving the  [Hardware Unit](#_bookmark25).  The number of the [Hardware Unit](#_bookmark25) 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](#_bookmark5) and  therefore refers all API services configured for the used [Hardware Unit](#_bookmark25)(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](#_bookmark24) #0 within the "A" Hardware Unit. |
| Hardware request | All [L-PDU](#_bookmark3) data will be set in the Hardware of i.e. [CAN](#_bookmark24)  [Controller](#_bookmark24) #0 within Hardware Unit "A" and the transmit request enabled. |
| [CanIf\_Transmit](#_bookmark279) [(PduId\_2,](#_bookmark279) [PduInfoPtr\_2)](#_bookmark279) | Upper layer initiates [Transmit Request](#_bookmark33). The PduId leads to  another [CAN Controller](#_bookmark24) and then to another [Hardware](#_bookmark25) [Unit](#_bookmark25).  The number of the [Hardware Unit](#_bookmark25) 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](#_bookmark5) and therefore refers all API services configured for the used [Hardware Unit](#_bookmark25)(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](#_bookmark24) #1 within the "B" Hardware Unit. |
| Hardware request | All [L-PDU](#_bookmark3) data will be set in the Hardware of i.e. [CAN](#_bookmark24)  [Controller](#_bookmark24) #1 within Hardware Unit "B" and the transmit request enabled. |

##### Notification mechanism using multiple CAN Drivers

Even if multiple [CanDrvs](#_bookmark5) are used in a single ECU Every notification callback service invoked by [CanDrvs](#_bookmark5) at the [CanIf](#_bookmark8) exists only once. This means, that [CanIf](#_bookmark8) has to identify calling [CanDrv](#_bookmark5) using the passed parameters. [CanIf](#_bookmark8) identifies the calling [CanDrv](#_bookmark5) from the ControllerId within the Mailbox (Can\_HwType) structure.



CanIf User

Receive Interrupt()

CanIf\_RxIndication(const Can\_HwType\*, const PduInfoType\*)

<User\_RxIndication>(PduIdType, const PduInfoType\*)

Received L-PDU

validation check (SW Filtering, Data Length Check)

Copy Data()

Copy Data()

<User\_RxIndication>()

CanIf\_RxIndication()

Receive Interrupt()

Receive Interrupt()

CanIf\_RxIndication(const Can\_HwType\*, const PduInfoType\*)

Received L-PDU validation check (SW Filtering, Data Length Check)

<User\_RxIndication>(PduIdType, const PduInfoType\*)

Copy data()

Copy data()

<User\_RxIndication>()

CanIf\_RxIndication()

Receive Interrupt()

«Peripheral» CanController

Can\_99\_Ext2: Can

Can\_99\_Ext1: Can

«module» CanIf

**Figure 7.11: Receive interrupt with multiple** [**CanDrvs**](#_bookmark5) **- simplified**

|  |  |
| --- | --- |
| **Operations called** | **Description** |
| Receive Interrupt | [CAN Controller](#_bookmark24) 1 signals a successful reception and triggers a  *receive interrupt*. The *ISR* of [CanDrv](#_bookmark5) A is invoked. |
| [CanIf\_RxIndication](#_bookmark383) [(Mailbox\_1,](#_bookmark383) [PduInfoPtr\_1)](#_bookmark383) | The reception is indicated to [CanIf](#_bookmark8) by calling of  [CanIf\_RxIndication()](#_bookmark383). The pointer Mailbox\_1 identifies the [HRH](#_bookmark18) and its corresponding CAN Controller, which contains  the received [L-PDU](#_bookmark3) specified by PduInfoPtr\_1. |
| Validation check (SW Filter-  ing, Data Length Check) | The Software Filtering checks, whether the [Received L-PDU](#_bookmark26) will  be processed on a local ECU. If not, the Received [L-SDU](#_bookmark4) is not indicated to upper layers and further processing is suppressed. If the [L-PDU](#_bookmark3) is found, the Data Length of the [Received L-PDU](#_bookmark26) is compared with the expected, statically configured one for the  received [L-PDU](#_bookmark3). |
| <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](#_bookmark4). The second parameter is the reference on PduInfoType which provides access to the buffer containing the [L-SDU](#_bookmark4). |

|  |  |
| --- | --- |
| Receive Interrupt | The [CAN Controller](#_bookmark24) 2 signals a successful reception and  triggers a *receive interrupt*. The *ISR* of [CanDrv](#_bookmark5) B is invoked. |
| [CanIf\_RxIndication](#_bookmark383) [(Mailbox\_2,](#_bookmark383) [PduInfoPtr\_2)](#_bookmark383) | The reception is indicated to [CanIf](#_bookmark8) by calling of  [CanIf\_RxIndication()](#_bookmark383). The pointer Mailbox\_2 identifies the [HRH](#_bookmark18) and its corresponding CAN Controller, which contains  the received [L-PDU](#_bookmark3) specified by PduInfoPtr\_2. |
| Validation check (SW Filter-  ing, Data Length Check) | The Software Filtering checks, whether the [Received L-PDU](#_bookmark26) will  be processed on a local ECU. If not, the Received [L-SDU](#_bookmark4) is not indicated to upper layers and further processing is suppressed. If the [L-PDU](#_bookmark3) is found, the Data Length of the [Received L-PDU](#_bookmark26) is compared with the expected, statically configured one for the  received [L-PDU](#_bookmark3). |
| <User\_RxIndication> (CanRxPduId\_2, CanPduInfoPtr\_2) | The corresponding receive indication service of the upper layer is  called. This signals a successful reception to the target upper layer. The parameter CanRxPduId\_2 specifies the ID of the received [L-SDU](#_bookmark4). The second parameter is the reference on PduInfoType which provides access to the buffer containing the [L-SDU](#_bookmark4). |

#### Partial Networking

**[SWS\_CANIF\_00747]** *[*If *Partial Networking* (PN) is enabled (see [CanIfPublicPn-](#_bookmark466) [Support](#_bookmark466)), [CanIf](#_bookmark8) shall support a PnTxFilter per [CAN Controller](#_bookmark24) which overlays the *PDU channel modes*.*]()*

**[SWS\_CANIF\_00748]** *[*The PnTxFilter of [[SWS\_CANIF\_00747](#_bookmark230)] shall only have an effect and transition its modes (enabled/disabled) if more than zero [Tx L-PDUs](#_bookmark27) per [CAN Controller](#_bookmark24) are configured as CanIfTxPduPnFilterPdu (see [CanIfTx-](#_bookmark486) [PduPnFilterPdu](#_bookmark486)).*]()*

**[SWS\_CANIF\_00863]** *[*PnTxFilter shall be enabled during initialization (ref. to [[SWS\_CANIF\_00747](#_bookmark230)] and [[SWS\_CANIF\_00748](#_bookmark231)]).*]()*

**[SWS\_CANIF\_00749]** *[*If [CanIf\_SetControllerMode(ControllerId, CAN\_-](#_bookmark264) [CS\_SLEEP)](#_bookmark264) is called the PnTxFilter of the corresponding [CAN Controller](#_bookmark24) shall be enabled (ref. to [[SWS\_CANIF\_00748](#_bookmark231)] and [[SWS\_CANIF\_00747](#_bookmark230)]).*]()*

**[SWS\_CANIF\_00750]** *[*If the PnTxFilter of a [CAN Controller](#_bookmark24) is enabled, [CanIf](#_bookmark8) shall block all Tx requests to that [CAN Controller](#_bookmark24) (return E\_NOT\_OK when [CanIf\_-](#_bookmark279) [Transmit()](#_bookmark279) is called), except if the requested [Tx L-PDUs](#_bookmark27) is one of the configured CanIfTxPduPnFilterPdus of that [CAN Controller](#_bookmark24). These CanIfTxPduPnFil- terPdus shall always be passed to the corresponding [CAN Driver](#_bookmark5).*]()*

**[SWS\_CANIF\_00751]** *[*If [CanIf\_TxConfirmation()](#_bookmark377) is called, the corre- sponding PnTxFilter shall be disabled (ref. to [[SWS\_CANIF\_00747](#_bookmark230)] and [[SWS\_CANIF\_00748](#_bookmark231)]).*]()*

**[SWS\_CANIF\_00896]** *[*If [CanIf\_RxIndication()](#_bookmark383) is called and PnTxFilter is en- abled, the corresponding PnTxFilter shall be disabled (ref. to [[SWS\_CANIF\_00747](#_bookmark230)] and [[SWS\_CANIF\_00748](#_bookmark231)]).*]()*

**[SWS\_CANIF\_00752]** *[*If the PnTxFilter of a [CAN Controller](#_bookmark24) is disabled, [CanIf](#_bookmark8) shall behave as requested via [CanIf\_SetPduMode()](#_bookmark304) (see [[SWS\_CANIF\_00008](#_bookmark303)]).*] ()*

**[SWS\_CANIF\_00878]** *[*If [CanIf\_SetPduMode(ControllerId, CANIF\_TX\_OF-](#_bookmark304) [FLINE)](#_bookmark304) is called and Partial Networking is enabled (ref. to [CanIfPublicPnSup-](#_bookmark466) [port](#_bookmark466)) the PnTxFilter of the corresponding [CAN Controller](#_bookmark24) shall be enabled (ref. to [[SWS\_CANIF\_00748](#_bookmark231)] and [[SWS\_CANIF\_00747](#_bookmark230)]).*]()*

#### CAN FD Support

For performance reasons some [CAN Controllers](#_bookmark24) allow to use a Flexible Data-Rate feature called [CAN FD](#_bookmark6) (see [[12](#_bookmark47), ISO 11898-1:2015]). Besides, the higher baud rate for the payload [CAN FD](#_bookmark6) 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](#_bookmark6) support by the [CAN Controller](#_bookmark24) and if the [CAN Controller](#_bookmark24) is in [CAN FD](#_bookmark6) mode (valid CanControllerFdBaudrateConfig).

If an [L-SDU](#_bookmark4) shall be sent as [CAN FD](#_bookmark6) or conventional CAN 2.0 frame depends on the configured [CanIfTxPduCanIdType](#_bookmark484). [CanIf](#_bookmark8) indicates this to [CanDrv](#_bookmark5) utilizing the second most significant bit of PduInfo->id (Can\_IdType) passed while calling Can\_Write().

Note: If [CanDrv](#_bookmark5) is not in [CAN FD](#_bookmark6) mode (no CanControllerFdBaudrateConfig, the [L-PDU](#_bookmark3) will be sent as conventional CAN 2.0 frame as long as the SduLength <= 8 bytes.

Note: The arbitration phase of conventional CAN 2.0 frames and [CAN FD](#_bookmark6) frames does not differ if the same [CanId](#_bookmark7) is used. Therefore, even when using [CAN FD](#_bookmark6) frames each [CanId](#_bookmark7) must not be used more than once.

Which kind of frame was received by [CanDrv](#_bookmark5) is also indicated utilizing the second most significant bit of the Can\_IdType passed with [CanIf\_RxIndication()](#_bookmark383) (Mailbox-

->CanId). Based on this information [CanIf](#_bookmark8) decides how to map to the configured

[L-SDU](#_bookmark4) (CanIfRxPduCfg) as described in [[SWS\_CANIF\_00877](#_bookmark154)].

Note: If upper layers don’t care if a message was received by conventional CAN 2.0 frame or [CAN FD](#_bookmark6) frame, it is possible to use only one [CanIfRxPduCfg](#_bookmark495) for both types (see [CanIfRxPduCanIdType](#_bookmark498)). 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.

#### 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](#_bookmark67))*

**[SWS\_CANIF\_00913]** *[*If security event reporting has been enabled for the [CanIf](#_bookmark8) module ([CanIfEnableSecurityEventReporting](#_bookmark463) = true) the respective security events shall bereported to the IdsM via the interfaces defined in AUTOSAR\_SWS\_- BSWGeneral.*]([RS\_Ids\_00810](#_bookmark67))*

**[SWS\_CANIF\_00915]** *[*If [CanIf\_ErrorNotification()](#_bookmark411) is called by [CanDrv](#_bookmark5), the function shall evaluate whether a Tx related error was detected. If this is the case the [CanIf](#_bookmark8)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](#_bookmark67))*

**[SWS\_CANIF\_00916]** *[*If [CanIf\_ErrorNotification()](#_bookmark411) is called by [CanDrv](#_bookmark5), the function shall evaluate whether a Rx related error was detected. If this is the case the [CanIf](#_bookmark8) 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](#_bookmark67))*

**[SWS\_CANIF\_00917]** *[*If [CanIf\_ControllerErrorStatePassive()](#_bookmark407) is called by [CanDrv](#_bookmark5), the [CanIf](#_bookmark8) shall report the security event CANIF\_SEV\_ERRORSTATE\_PAS- SIVE 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](#_bookmark67))*

**[SWS\_CANIF\_00918]** *[*If CanIf\_ControllerBusOff is called by [CanDrv](#_bookmark5), the [CanIf](#_bookmark8) 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](#_bookmark67))*

#### Error classification

##### 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 |

##### Runtime Errors [SWS\_CANIF\_91007] *[*

|  |  |  |
| --- | --- | --- |
| ***Type of error*** | ***Related error code*** | ***Error value*** |
| Failed Data Length Check | CANIF\_E\_INVALID\_DATA\_LENGTH | 61 |
| Data Length | CANIF\_E\_DATA\_LENGTH\_MISMATCH | 62 |
| Transmit requested on offline PDU channel | CANIF\_E\_STOPPED | 70 |
| Message length was exceeding the maximum length | CANIF\_E\_TXPDU\_LENGTH\_EXCEEDED | 90 |

*]()*

##### Transient Faults

There are no transient faults.

##### Production Errors

There are no production errors.

##### Extended Production Errors

There are no extended production errors.

### API specification

#### Imported types

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

**[SWS\_CANIF\_00142] [**

|  |  |  |
| --- | --- | --- |
| ***Module*** | ***Header File*** | ***Imported Type*** |
| Can | Can\_GeneralTypes.h | Can\_ControllerStateType |
| Can\_GeneralTypes.h | Can\_ErrorStateType |
| Can\_GeneralTypes.h | Can\_ErrorType |
| Can\_GeneralTypes.h | Can\_HwHandleType |
| Can\_GeneralTypes.h | Can\_HwType |
| Can\_GeneralTypes.h | Can\_IdType |
| Can\_GeneralTypes.h | Can\_PduType |
| Can\_GeneralTypes.h | Can\_TimeStampType (draft) |
| CanTrcv | Can\_GeneralTypes.h | CanTrcv\_TrcvModeType |
| Can\_GeneralTypes.h | CanTrcv\_TrcvWakeupModeType |
| Can\_GeneralTypes.h | CanTrcv\_TrcvWakeupReasonType |
| 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](#_bookmark89),* [*SRS\_BSW\_00353*](#_bookmark90)*,* [*SRS\_BSW\_00361*](#_bookmark92)*)*

#### Type definitions

##### CanIf\_ConfigType

[SWS\_CANIF\_00144] *[*

|  |  |  |
| --- | --- | --- |
| ***Name*** | CanIf\_ConfigType | |
| ***Kind*** | Structure | |
| ***Elements*** | implementation specific | |
| ***Type*** | – |
| ***Comment*** | The contents of the initialization data structure are CAN interface specific |

|  |  |
| --- | --- |
| ***Description*** | This type defines a data structure for the post build parameters of the CAN interface for all underlying CAN drivers. At initialization the CanIf gets a pointer to a structure of this type to get access to its configuration data, which is necessary for initialization. |
| ***Available via*** | CanIf.h |

**[SWS\_CANIF\_00523]** *[*The initialization data structure for a specific CanIf\_Config- Type shall include the definition of [CanIf](#_bookmark8) public parameters and the definition for each [L-PDU](#_bookmark3)/[L-SDU](#_bookmark4).*]()*

Note: The definition of [CanIf](#_bookmark8) public parameters and the definition for each [L-PDU](#_bookmark3)/ [L-SDU](#_bookmark4) are specified in [chapter](#_bookmark451) [10](#_bookmark451).

##### CanIf\_PduModeType [SWS\_CANIF\_00137] *[*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name*** | CanIf\_PduModeType | | |
| ***Kind*** | Enumeration | | |
| ***Range*** | CANIF\_OFFLINE | 0x00 | = 0 Transmit and receive path of the corresponding channel are disabled => no communication mode |
| CANIF\_TX\_OFFLINE | 0x01 | Transmit path of the corresponding channel is disabled. The receive path is enabled. |
| CANIF\_TX\_OFFLINE\_ ACTIVE | 0x02 | Transmit path of the corresponding channel is in offline active mode (see SWS\_ CANIF\_00072). The receive path is enabled. This mode requires CanIfTxOfflineActive Support = TRUE. |
| CANIF\_ONLINE | 0x03 | Transmit and receive path of the corresponding channel are enabled => full operation mode |
| ***Description*** | The PduMode of a channel defines its transmit or receive activity. Communication direction (transmission and/or reception) of the channel can be controlled separately or together by upper layers. | | |
| ***Available via*** | CanIf.h | | |

##### CanIf\_NotifStatusType [SWS\_CANIF\_00201] *[*

|  |  |
| --- | --- |
| ***Name*** | CanIf\_NotifStatusType |
| ***Kind*** | Enumeration |

*q*

*Δ*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Range*** | CANIF\_TX\_RX\_ NOTIFICATION | – | The requested Rx/Tx CAN L-PDU was successfully transmitted or received. |
| CANIF\_NO\_NOTIFICATION | 0x00 | No transmit or receive event occurred for the requested L-PDU. |
| ***Description*** | Return value of CAN L-PDU notification status. | | |
| ***Available via*** | CanIf.h | | |

*]()*

#### Function definitions

**[SWS\_CANIF\_00661]** *[*All CanIf API services other than [CanIf\_Init()](#_bookmark257) and [CanIf\_GetVersionInfo()](#_bookmark314) shall not execute their normal operation and return E\_- NOT\_OK unless the [CanIf](#_bookmark8) has been initialized with a preceding call of [CanIf\_Init](#_bookmark257) [()](#_bookmark257).*]()*

##### CanIf\_Init [SWS\_CANIF\_00001] *[*

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

*](*[*SRS\_BSW\_00405*](#_bookmark95)*,* [*SRS\_BSW\_00101*](#_bookmark70)*,* [*SRS\_BSW\_00358*](#_bookmark91)*,* [*SRS\_BSW\_00414*](#_bookmark98)*,* [*SRS\_Can\_01021*](#_bookmark119)*,* [*SRS\_Can\_01022*](#_bookmark120)*)*

Note: All underlying CAN controllers and transceivers still remain not operational. Note: The service [CanIf\_Init()](#_bookmark257) is called only by the [EcuM](#_bookmark16).

**[SWS\_CANIF\_00085]** *[*The service [CanIf\_Init()](#_bookmark257) shall initialize the global variables and data structures of the [CanIf](#_bookmark8) including flags and buffers.*]()*

##### CanIf\_DeInit [SWS\_CANIF\_91002] *[*

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

*]([SRS\_Can\_01168](#_bookmark136),* [*SRS\_BSW\_00336*](#_bookmark87)*)*

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()](#_bookmark261) function has to be sure there are no on-going transmissions/receptions, nor any pending transmission confirmations.

##### CanIf\_SetControllerMode

[SWS\_CANIF\_00003] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_SetControllerMode | |
| ***Syntax*** | Std\_ReturnType CanIf\_SetControllerMode **(** uint8 ControllerId, Can\_ControllerStateType ControllerMode  **)** | |
| ***Service ID [hex]*** | 0x03 | |
| ***Sync/Async*** | Asynchronous | |
| ***Reentrancy*** | Reentrant (Not for the same controller) | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for mode transition. |
| ControllerMode | Requested mode transition |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Controller mode request has been accepted E\_NOT\_OK: Controller mode request has not been accepted |
| ***Description*** | This service calls the corresponding CAN Driver service for changing of the CAN controller mode. | |
| Available via | CanIf.h | |

*]([SRS\_Can\_01027](#_bookmark121))*

Note: The service [CanIf\_SetControllerMode()](#_bookmark264) initiates a transition to the re- quested CAN controller mode ControllerMode of the CAN controller which is as- signed by parameter ControllerId.

**[SWS\_CANIF\_00308]** *[*The service [CanIf\_SetControllerMode()](#_bookmark264) shall call Can\_SetControllerMode(Controller, Transition) for the requested CAN controller.*]()*

**[SWS\_CANIF\_00311]** *[*If parameter ControllerId of [CanIf\_SetController-](#_bookmark264) [Mode()](#_bookmark264) 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()](#_bookmark264) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00774]** *[*If parameter ControllerMode of [CanIf\_SetController-](#_bookmark264) [Mode()](#_bookmark264) has an invalid value (not CAN\_CS\_STARTED, CAN\_CS\_SLEEP or CAN\_CS\_- STOPPED), the CanIfshall report development error code CANIF\_E\_PARAM\_CTRLMODE to the Det\_ReportError service of the DET module, when [CanIf\_SetCon-](#_bookmark264) [trollerMode()](#_bookmark264) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

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

##### CanIf\_GetControllerMode [SWS\_CANIF\_00229] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_GetControllerMode | |
| ***Syntax*** | Std\_ReturnType CanIf\_GetControllerMode **(** uint8 ControllerId, Can\_ControllerStateType**\*** ControllerModePtr  **)** | |
| ***Service ID [hex]*** | 0x04 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for current operation mode. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | ControllerModePtr | Pointer to a memory location, where the current mode of the CAN controller will be stored. |
| ***Return value*** | Std\_ReturnType | E\_OK: Controller mode request has been accepted. E\_NOT\_OK: Controller mode request has not been accepted. |
| ***Description*** | This service calls the corresponding CAN Driver service for obtaining the current status of the CAN controller. | |

*]([SRS\_Can\_01028](#_bookmark122))*

**[SWS\_CANIF\_00313]** *[*If parameter ControllerId of [CanIf\_GetController-](#_bookmark269) [Mode()](#_bookmark269) 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()](#_bookmark269) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00656]** *[*If parameter ControllerModePtr of [CanIf\_GetCon-](#_bookmark269) [trollerMode()](#_bookmark269) 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()](#_bookmark269) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: The ID of the CAN controller module is published inside the configuration de- scription of the CanIf.

##### CanIf\_GetControllerErrorState [SWS\_CANIF\_91001] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_GetControllerErrorState | |
| ***Syntax*** | Std\_ReturnType CanIf\_GetControllerErrorState **(**  uint8 ControllerId, Can\_ErrorStateType**\*** ErrorStatePtr  **)** | |
| ***Service ID [hex]*** | 0x4b | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant for the same ControllerId | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | ErrorStatePtr | Pointer to a memory location, where the error state of the CAN controller will be stored. |
| ***Return value*** | Std\_ReturnType | E\_OK: Error state request has been accepted. E\_NOT\_OK: Error state request has not been accepted. |
| ***Description*** | This service calls the corresponding CAN Driver service for obtaining the error state of the CAN controller. | |
| ***Available via*** | CanIf.h | |

*]([SRS\_Can\_01169](#_bookmark137))*

**[SWS\_CANIF\_00898]** *[*If parameter ControllerId of [CanIf\_GetCon-](#_bookmark274) [trollerErrorState()](#_bookmark274) has an invalid value, the CanIf shall report develop- ment error code CANIF\_E\_PARAM\_CONTROLLERID to the Det\_ReportError service of the DET, when [CanIf\_GetControllerErrorState()](#_bookmark274) is called.*] (*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00899]** *[*If parameter ErrorStatePtr of [CanIf\_GetCon-](#_bookmark274) [trollerErrorState()](#_bookmark274) 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()](#_bookmark274) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

##### CanIf\_Transmit [SWS\_CANIF\_00005] *[*

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

*]([SRS\_Can\_01008](#_bookmark113))*

Note: The corresponding [CAN Controller](#_bookmark24) and [HTH](#_bookmark19) have to be resolved by the Tx- PduId.

**[SWS\_CANIF\_00317]** *[*The service [CanIf\_Transmit()](#_bookmark279) 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()](#_bookmark279) shall call Can\_Write() with the hard- ware transmit handle corresponding to the provided TxPduId and a Can\_PduType structure where:

* swPduHandle is set to the CanTxPduId used in the corresponding [CanIf\_-](#_bookmark377) [TxConfirmation()](#_bookmark377) call
* length is set to the value provided as PduInfoPtr->SduLength, possibly reduced according to [[SWS\_CANIF\_00894](#_bookmark286)]
* 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](#_bookmark4) user memory, *CAN Identifier*, [L-SDU](#_bookmark4) han- dle and Data Length (see [[1](#_bookmark36), Specification of CAN Driver]).

**[SWS\_CANIF\_00243]** *[*[CanIf](#_bookmark8) shall set the two most significant bits (’IDentifier Ex- tension flag’ (see [[12](#_bookmark47), ISO 11898-1:2015]) and ’CAN FD flag’) of the *CanId* ( PduInfoPtr->id) before [CanIf](#_bookmark8) passes the predefined *CanId* to [CanDrv](#_bookmark5) at call of Can\_Write() (see [[1](#_bookmark36), Specification of CAN Driver], definition of Can\_IdType [SWS\_Can\_00416]). The *CanId* format type of each [CAN L-PDU](#_bookmark3) can be configured by [CanIfTxPduCanIdType](#_bookmark484), refer to [CanIfTxPduCanIdType](#_bookmark484).*](*[*SRS\_Can\_01141*](#_bookmark133)*)*

**[SWS\_CANIF\_00882]** *[*[CanIf\_Transmit()](#_bookmark279) shall accept a NULL pointer as PduIn- foPtr->SduDataPtr, if the PDU is configured for triggered transmission: CanIfTx- PduTriggerTransmit = TRUE.*]()*

**[SWS\_CANIF\_00162]** *[*If the call of Can\_Write() returns E\_OK the transmit request service [CanIf\_Transmit()](#_bookmark279) shall return E\_OK.*]()*

Note: If the call of Can\_Write() returns E\_NOT\_OK, then the transmit request service [CanIf\_Transmit()](#_bookmark279) shall return E\_NOT\_OK. If the transmit request service [CanIf\_-](#_bookmark279) [Transmit()](#_bookmark279) 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()](#_bookmark279) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_INVALID\_TXPDUID to the Det\_ReportError service of the DET, when [CanIf\_Transmit()](#_bookmark279) is called.*] (*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00320]** *[*If parameter PduInfoPtr of [CanIf\_Transmit()](#_bookmark279) has an in- valid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_POINTER to the Det\_ReportError service of the DET module, when [CanIf\_Transmit()](#_bookmark279) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00893]** *[*When [CanIf\_Transmit()](#_bookmark279) is called with PduInfoPtr->Sd- uLength 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

[CanIf](#_bookmark8) shall report runtime error code CANIF\_E\_DATA\_LENGTH\_MISMATCH to the

Det\_ReportRuntimeError() service of the *DET*.*]()*

Note: Besides static configured transmissions there are dynamic transmissions, too. Therefore, the valid data length is always passed by PduInfoPtr->SduLength. Furthermore, even the frame type might change via [CanIf\_SetDynamicTxId()](#_bookmark317). [[SWS\_CANIF\_00893](#_bookmark285)] ensures that not matching transmit requests can be detected via DET.

**[SWS\_CANIF\_00894]** *[*When [CanIf\_Transmit()](#_bookmark279) is called with PduInfoPtr->Sd- uLength exceeding the length of the global PDU (see [ECUC\_EcuC\_00078]) refer- enced by TxPduId and CanIfTxPduTruncation is enabled, [CanIf](#_bookmark8) 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()](#_bookmark279) is called with PduInfoPtr->Sd- uLength exceeding the length of the global PDU (see [ECUC\_EcuC\_00078]) refer- enced by TxPduId and CanIfTxPduTruncation is disabled, [CanIf](#_bookmark8) shall report the runtime error CANIF\_E\_TXPDU\_LENGTH\_EXCEEDED and return E\_NOT\_OK with- out further actions.*]()*

Note: During the call of [CanIf\_Transmit()](#_bookmark279) the buffer of PduInfoPtr is controlled by [CanIf](#_bookmark8) and this buffer should not be accessed for read/write from another call con- text. After return of this call the ownership changes to the upper layer.

##### CanIf\_ReadRxPduData [SWS\_CANIF\_00194] *[*

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

*]([SRS\_Can\_01125](#_bookmark125),* [*SRS\_Can\_01129*](#_bookmark127)*)*

**[SWS\_CANIF\_00324]** *[*The function [CanIf\_ReadRxPduData()](#_bookmark289) shall not accept a request and return E\_NOT\_OK, if the corresponding controller mode refrenced by Con- trollerId 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()](#_bookmark289) has an invalid value, e.g. not configured to be stored within [CanIf](#_bookmark8) via [CanIfRxP-](#_bookmark502) [duReadData](#_bookmark502), [CanIf](#_bookmark8) shall report development error code CANIF\_E\_INVALID\_RX- PDUID to the Det\_ReportError service of the DET, when [CanIf\_ReadRxPdu-](#_bookmark289) [Data()](#_bookmark289) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00326]** *[*If parameter CanIfRxInfoPtr of [CanIf\_ReadRxPduData](#_bookmark289) [()](#_bookmark289) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_- PARAM\_POINTER to the Det\_ReportError service of the DET module, when [CanIf\_ReadRxPduData()](#_bookmark289) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00329]** *[*[CanIf\_ReadRxPduData()](#_bookmark289) shall not be used for CanIfRxS- duId, which are defined to receive multiple CAN-Ids (range reception).*]()*

Note: During the call of [CanIf\_ReadRxPduData()](#_bookmark289) the buffer of CanIfRxInfoPtr is controlled by [CanIf](#_bookmark8) 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()](#_bookmark289): This API can be enabled or disabled at pre-compile time configuration by the configuration parameter [CanIfPublicReadRxPduDataApi](#_bookmark467).*]()*

##### CanIf\_ReadTxNotifStatus [SWS\_CANIF\_00202] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ReadTxNotifStatus | |
| ***Syntax*** | CanIf\_NotifStatusType CanIf\_ReadTxNotifStatus **(**  PduIdType CanIfTxSduId  **)** | |
| ***Service ID [hex]*** | 0x07 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | CanIfTxSduId | L-SDU handle to be transmitted. This handle specifies the corresponding CAN L-SDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | [CanIf\_NotifStatusType](#_bookmark253) | Current confirmation status of the corresponding CAN Tx L-PDU. |
| ***Description*** | This service returns the confirmation status (confirmation occurred or not) of a specific static or dynamic CAN Tx L-PDU, requested by the CanIfTxSduId. | |
| ***Available via*** | CanIf.h | |

*]([SRS\_Can\_01130](#_bookmark128))*

Note: This function notifies the upper layer about any transmit confirmation event to the corresponding requested [L-SDU](#_bookmark4).

**[SWS\_CANIF\_00393]** *[*If configuration parameters [CanIfPublicReadTxPduNoti-](#_bookmark469) [fyStatusApi](#_bookmark469) and [CanIfTxPduReadNotifyStatus](#_bookmark487) for the transmitted [L-SDU](#_bookmark4) are set to TRUE, and if [CanIf\_ReadTxNotifStatus()](#_bookmark294) is called, the [CanIf](#_bookmark8) shall reset the notification status for the transmitted [L-SDU](#_bookmark4).*]()*

**[SWS\_CANIF\_00331]** *[*If parameter CanIfTxSduId of [CanIf\_ReadTxNotifSta-](#_bookmark294) [tus()](#_bookmark294) is out of range or if no status information was configured for this CAN Tx [L-SDU](#_bookmark4), [CanIf](#_bookmark8) shall report development error code CANIF\_E\_INVALID\_TXPDUID to the Det\_ReportError service of the DET when [CanIf\_ReadTxNotifStatus](#_bookmark294) [()](#_bookmark294) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00335]** *[*Configuration of CanIf\_ReadTxNotifyStatus(): This API can be enabled or disabled at pre-compile time configuration globally by the parameter [CanIfPublicReadTxPduNotifyStatusApi](#_bookmark469).*]()*

##### CanIf\_ReadRxNotifStatus [SWS\_CANIF\_00230] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ReadRxNotifStatus | |
| ***Syntax*** | CanIf\_NotifStatusType CanIf\_ReadRxNotifStatus **(**  PduIdType CanIfRxSduId  **)** | |
| ***Service ID [hex]*** | 0x08 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | CanIfRxSduId | Receive L-SDU handle specifying the corresponding CAN L-SDU ID and implicitly the CAN Driver instance as well as the corresponding CAN controller device. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | [CanIf\_NotifStatusType](#_bookmark253) | Current indication status of the corresponding CAN Rx L-PDU. |
| ***Description*** | This service returns the indication status (indication occurred or not) of a specific CAN Rx L-PDU, requested by the CanIfRxSduId. | |
| ***Available via*** | CanIf.h | |

*]([SRS\_Can\_01130](#_bookmark128),* [*SRS\_Can\_01131*](#_bookmark129)*)*

Note: This function notifies the upper layer about any receive indication event to the corresponding requested [L-SDU](#_bookmark4).

**[SWS\_CANIF\_00394]** *[*If configuration parameters [CanIfPublicReadRxPduNoti-](#_bookmark468) [fyStatusApi](#_bookmark468) and [CanIfRxPduReadNotifyStatus](#_bookmark503) are set to TRUE, and if [CanIf\_ReadRxNotifStatus()](#_bookmark299) is called, then [CanIf](#_bookmark8) shall reset the notification sta- tus for the received [L-SDU](#_bookmark4).*]()*

**[SWS\_CANIF\_00336]** *[*If parameter CanIfRxSduId of [CanIf\_ReadRxNotifSta-](#_bookmark299) [tus()](#_bookmark299) is out of range or if status for CanRxPduId was requested whereas [CanIfRx-](#_bookmark502) [PduReadData](#_bookmark502) is disabled or if no status information was configured for this CAN Rx

[L-SDU](#_bookmark4), [CanIf](#_bookmark8) shall report development error code CANIF\_E\_INVALID\_RXPDUID to the Det\_ReportError service of the DET, when [CanIf\_ReadRxNotifStatus](#_bookmark299) [()](#_bookmark299) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: The function [CanIf\_ReadRxNotifStatus()](#_bookmark299) must not be used for CanI- fRxSduIds, which are defined to receive multiple CAN-Ids (range reception).

**[SWS\_CANIF\_00340]** *[*Configuration of [CanIf\_ReadRxNotifStatus()](#_bookmark299): This API can be enabled or disabled at pre-compile time configuration globally by the parameter [CanIfPublicReadRxPduNotifyStatusApi](#_bookmark468).*]()*

##### CanIf\_SetPduMode [SWS\_CANIF\_00008] *[*

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

*]()*

Note: The channel parameter denoting the predefined logical PDU channel can be derived from parameter ControllerId of function [CanIf\_SetPduMode()](#_bookmark304).

**[SWS\_CANIF\_00341]** *[*If [CanIf\_SetPduMode()](#_bookmark304) is called with invalid Control- lerId, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROL- LERID to the Det\_ReportError service of the DET module.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00860]** *[*If [CanIf\_SetPduMode()](#_bookmark304) is called with invalid PduMod- eRequest, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_PDU\_MODE to the Det\_ReportError service of the *DET* module.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00874]** *[*The service [CanIf\_SetPduMode()](#_bookmark304) shall not accept any re- quest and shall return E\_NOT\_OK, if the controller mode referenced by ControllerId is not in state CAN\_CS\_STARTED.*]()*

##### CanIf\_GetPduMode [SWS\_CANIF\_00009] *[*

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

**[SWS\_CANIF\_00346]** *[*If [CanIf\_GetPduMode()](#_bookmark309) is called with invalid Control- lerId, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROL- LERID to the Det\_ReportError service of the DET module.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00657]** *[*If [CanIf\_GetPduMode()](#_bookmark309) is called with invalid PduModePtr, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_POINTER to the Det\_- ReportError service of the DET module.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

##### CanIf\_GetVersionInfo [SWS\_CANIF\_00158] *[*

|  |  |
| --- | --- |
| ***Service Name*** | CanIf\_GetVersionInfo |
| ***Syntax*** | void CanIf\_GetVersionInfo **(**  Std\_VersionInfoType**\*** VersionInfo  **)** |

*q*

*Δ*

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

*]([SRS\_BSW\_00407](#_bookmark96),* [*SRS\_BSW\_00411*](#_bookmark97)*)*

##### CanIf\_SetDynamicTxId [SWS\_CANIF\_00189] *[*

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

*]()*

Note: [CanIf\_SetDynamicTxId()](#_bookmark317) may be interrupted by [CanIf\_Transmit()](#_bookmark279) called by several modules in the communication stack. Therefore precautions for preventing inconsistency need to be considered.

**[SWS\_CANIF\_00352]** *[*If parameter CanIfTxSduId of [CanIf\_SetDynamicTxId](#_bookmark317) [()](#_bookmark317) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_IN- VALID\_TXPDUID to the Det\_ReportError service of the DET module, when [CanIf\_SetDynamicTxId()](#_bookmark317) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00353]** *[*If parameter CanId of [CanIf\_SetDynamicTxId()](#_bookmark317) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CANID to the Det\_ReportError service of the DET module, when [CanIf\_SetDynamic-](#_bookmark317) [TxId()](#_bookmark317) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00355]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_SetDynam-](#_bookmark317) [icTxId()](#_bookmark317), then the function [CanIf\_SetDynamicTxId()](#_bookmark317) shall not execute a recon- figuration of Tx CanId.*]()*

**[SWS\_CANIF\_00357]** *[*Configuration of [CanIf\_SetDynamicTxId()](#_bookmark317): This function shall be pre compile time configurable On/Off by the configuration parameter [CanIf-](#_bookmark470) [PublicSetDynamicTxIdApi](#_bookmark470).*]()*

##### CanIf\_SetTrcvMode [SWS\_CANIF\_00287] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_SetTrcvMode | |
| ***Syntax*** | Std\_ReturnType CanIf\_SetTrcvMode **(** uint8 TransceiverId, CanTrcv\_TrcvModeType TransceiverMode  **)** | |
| ***Service ID [hex]*** | 0x0d | |
| ***Sync/Async*** | Asynchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for mode transition |
| TransceiverMode | Requested mode transition |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Transceiver mode request has been accepted. E\_NOT\_OK: Transceiver mode request has not been accepted. |
| ***Description*** | This service changes the operation mode of the tansceiver TransceiverId, via calling the corresponding CAN Transceiver Driver service. | |
| ***Available via*** | CanIf.h | |

*]()*

Note: For more details, please refer to the [[2](#_bookmark37), Specification of CAN Transceiver Driver].

**[SWS\_CANIF\_00358]** *[*The function [CanIf\_SetTrcvMode()](#_bookmark322) shall call the function CanTrcv\_SetOpMode(Transceiver, OpMode) on the corresponding requested CAN Transceiver Driver module.*]()*

Note: The parameters of the service CanTrcv\_SetOpMode() are of type:

* OpMode: CanTrcv\_TrcvModeType(desired operation mode)
* Transceiver: uint8 (Transceiver to which function call has to be applied)

(see [[2](#_bookmark37), Specification of CAN Transceiver Driver])

**[SWS\_CANIF\_00538]** *[*If parameter TransceiverId of [CanIf\_SetTrcvMode()](#_bookmark322) 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](#_bookmark322) [()](#_bookmark322) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

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](#_bookmark37)]). 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](#_bookmark37)]). But this is not checked by the CanIf.

**[SWS\_CANIF\_00648]** *[*If parameter TransceiverMode of [CanIf\_SetTrcvMode()](#_bookmark322) 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()](#_bookmark322) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: The function [CanIf\_SetTrcvMode()](#_bookmark322) should be applicable to all CAN transceivers with all values of TransceiverMode independent, if the transceiver hard- ware 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()](#_bookmark322): The number of supported transceiver types for each network is set up in the configuration phase (see [CanIfTrcvCfg](#_bookmark534) and [CanIfTrcvDrvCfg](#_bookmark532)). 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.*]()*

##### CanIf\_GetTrcvMode [SWS\_CANIF\_00288] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_GetTrcvMode | |
| ***Syntax*** | Std\_ReturnType CanIf\_GetTrcvMode **(**  uint8 TransceiverId, CanTrcv\_TrcvModeType**\*** TransceiverModePtr  **)** | |
| ***Service ID [hex]*** | 0x0e | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for current operation mode. |
| ***Parameters (inout)*** | None | |

*q*

*Δ*

|  |  |  |
| --- | --- | --- |
| ***Parameters (out)*** | TransceiverModePtr | Requested mode of requested network the Transceiver is connected to. |
| ***Return value*** | Std\_ReturnType | E\_OK: Transceiver mode request has been accepted. E\_NOT\_OK: Transceiver mode request has not been accepted. |
| ***Description*** | This function invokes CanTrcv\_GetOpMode and updates the parameter TransceiverModePtr with the value OpMode provided by CanTrcv. | |
| ***Available via*** | CanIf.h | |

*]()*

Note: For more details, please refer to the [[2](#_bookmark37), Specification of CAN Transceiver Driver].

**[SWS\_CANIF\_00363]** *[*The function [CanIf\_GetTrcvMode()](#_bookmark326) 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](#_bookmark37), Specification of CAN Transceiver Driver])

**[SWS\_CANIF\_00364]** *[*If parameter TransceiverId of [CanIf\_GetTrcvMode()](#_bookmark326) 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\_GetTr-](#_bookmark326) [cvMode()](#_bookmark326) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00650]** *[*If parameter TransceiverModePtr of [CanIf\_GetTrcv-](#_bookmark326) [Mode()](#_bookmark326) 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()](#_bookmark326) was called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00367]** *[*Configuration of [CanIf\_GetTrcvMode()](#_bookmark326): The number of supported transceiver types for each network is set up in the configuration phase (see [CanIfTrcvCfg](#_bookmark534) and [CanIfTrcvDrvCfg](#_bookmark532)). 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.*]()*

##### CanIf\_GetTrcvWakeupReason [SWS\_CANIF\_00289] *[*

|  |  |
| --- | --- |
| ***Service Name*** | CanIf\_GetTrcvWakeupReason |

*q*

*Δ*

|  |  |  |
| --- | --- | --- |
| ***Syntax*** | Std\_ReturnType CanIf\_GetTrcvWakeupReason **(** uint8 TransceiverId, CanTrcv\_TrcvWakeupReasonType**\*** TrcvWuReasonPtr  **)** | |
| ***Service ID [hex]*** | 0x0f | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for wake up reason. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | TrcvWuReasonPtr | provided pointer to where the requested transceiver wake up reason shall be returned |
| ***Return value*** | Std\_ReturnType | E\_OK: Transceiver wake up reason request has been accepted. E\_NOT\_OK: Transceiver wake up reason request has not been accepted. |
| ***Description*** | This service returns the reason for the wake up of the transceiver TransceiverId, via calling the corresponding CAN Transceiver Driver service. | |
| ***Available via*** | CanIf.h | |

*]()*

Note: The ability to detect and differentiate the possible wake up reasons depends strongly on the CAN transceiver hardware. For more details, please refer to the [[2](#_bookmark37), Specification of CAN Transceiver Driver].

**[SWS\_CANIF\_00368]** *[*The function [CanIf\_GetTrcvWakeupReason()](#_bookmark330) shall call CanTrcv\_GetBusWuReason(Transceiver, Reason) on the corresponding re- quested [CanTrcv](#_bookmark12).*]()*

Note: The parameters of the function CanTrcv\_GetBusWuReason() are of type:

* Reason: CanTrcv\_TrcvWakeupReasonType
* Transceiver: uint8 (Transceiver to which API call has to be applied) (see [[2](#_bookmark37), Specification of CAN Transceiver Driver])

**[SWS\_CANIF\_00537]** *[*If parameter TransceiverId of [CanIf\_GetTrcvWake-](#_bookmark330) [upReason()](#_bookmark330) 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()](#_bookmark330) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00649]** *[*If parameter TrcvWuReasonPtr of [CanIf\_GetTrcvWake-](#_bookmark330) [upReason()](#_bookmark330) has an invalid value, the CanIf shall report development error code CANIF\_E\_PARAM\_POINTER to the Det\_ReportError service of the DET mod- ule, when [CanIf\_GetTrcvWakeupReason()](#_bookmark330) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

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()](#_bookmark330) has a "per network" view and does not vote the more important reason or sequence internally. The same may be true if

e.g. one transceiver controls the power supply and the other is just powered or un- powered. Then one may be able to return 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()](#_bookmark330): The number of supported transceiver types for each network is set up in the configura- tion phase (see [CanIfTrcvCfg](#_bookmark534) and [CanIfTrcvDrvCfg](#_bookmark532)). 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.*]()*

##### CanIf\_SetTrcvWakeupMode [SWS\_CANIF\_00290] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_SetTrcvWakeupMode | |
| ***Syntax*** | Std\_ReturnType CanIf\_SetTrcvWakeupMode **(** uint8 TransceiverId, CanTrcv\_TrcvWakeupModeType TrcvWakeupMode  **)** | |
| ***Service ID [hex]*** | 0x10 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstracted CanIf TransceiverId, which is assigned to a CAN transceiver, which is requested for wake up notification mode transition. |
| TrcvWakeupMode | Requested transceiver wake up notification mode |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Will be returned, if the wake up notifications state has been changed to the requested mode.  E\_NOT\_OK: Will be returned, if the wake up notifications state change has failed or the parameter is out of the allowed range. The previous state has not been changed. |
| ***Description*** | This function shall call CanTrcv\_SetTrcvWakeupMode. | |
| ***Available via*** | CanIf.h | |

Note: For more details, please refer to [[2](#_bookmark37), Specification of CAN Transceiver Driver].

**[SWS\_CANIF\_00372]** *[*The function [CanIf\_SetTrcvWakeupMode()](#_bookmark334) shall call CanTrcv\_SetWakeupMode(Transceiver, TrcvWakeupMode) on the corre- sponding requested [CanTrcv](#_bookmark12).*]()*

Info: The parameters of the function CanTrcv\_SetWakeupMode() are of type:

* TrcvWakeupMode: CanTrcv\_TrcvWakeupModeType (see [[2](#_bookmark37), Specification of CAN Transceiver Driver])
* Transceiver: uint8 (Transceiver to which API call has to be applied) (see [[2](#_bookmark37), Specification of CAN Transceiver Driver])

Note: The following three paragraphs are already described in the Specification of CanTrcv (see [[2](#_bookmark37)]). They describe the behavior of a [CanTrcv](#_bookmark12) in the respective transceiver wake-up mode, which is requested in parameter TrcvWakeupMode.

CANIF\_TRCV\_WU\_ENABLE:

If the [CanTrcv](#_bookmark12) has a stored wake-up event pending for the addressed CanNetwork, the notification is executed within or immediately after the function CanTrcv\_SetTr- cvWakeupMode() (depending on the implementation).

CANIF\_TRCV\_WU\_DISABLE:

No notifications for wake-up events for the addressed CanNetwork are passed through the [CanTrcv](#_bookmark12). The transceiver device and the underlying communication driver has to buffer detected wake-up events and raise the event(s), when the wake-up noti- fication 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\_Set-](#_bookmark334) [TrcvWakeupMode()](#_bookmark334) with parameter CANIF\_TRCV\_WU\_CLEAR clears these bufferd events. Clearing of wake-up events has to be used, when the wake-up notification is disabled to clear all stored wake-up events under control of the higher layers of the [CanTrcv](#_bookmark12).

**[SWS\_CANIF\_00535]** *[*If parameter TransceiverId of [CanIf\_SetTrcvWakeup-](#_bookmark334) [Mode()](#_bookmark334) 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()](#_bookmark334) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00536]** *[*If parameter TrcvWakeupMode of [CanIf\_SetTrcvWakeup-](#_bookmark334) [Mode()](#_bookmark334) has an invalid value, the CanIf shall report development error code CANIF\_- E\_PARAM\_TRCVWAKEUPMODE to the Det\_ReportError service of the DET mod- ule, when [CanIf\_SetTrcvWakeupMode()](#_bookmark334) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00373]** *[*Configuration of [CanIf\_SetTrcvWakeupMode()](#_bookmark334): The num- ber of supported transceiver types for each network is set up in the configuration phase (see [CanIfTrcvCfg](#_bookmark534) and [CanIfTrcvDrvCfg](#_bookmark532)). 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.*]()*

##### CanIf\_CheckWakeup [SWS\_CANIF\_00219] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_CheckWakeup | |
| ***Syntax*** | Std\_ReturnType CanIf\_CheckWakeup **(**  EcuM\_WakeupSourceType WakeupSource  **)** | |
| ***Service ID [hex]*** | 0x11 | |
| ***Sync/Async*** | Asynchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | WakeupSource | Source device, which initiated the wake up event: CAN controller or CAN transceiver |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Will be returned, if the check wake up request has been accepted  E\_NOT\_OK: Will be returned, if the check wake up request has not been accepted |
| ***Description*** | This service checks, whether an underlying CAN driver or a CAN transceiver driver already signals a wakeup event. | |
| ***Available via*** | CanIf.h | |

*]()*

Note: *Integration Code* calls this function

**[SWS\_CANIF\_00398]** *[*If parameter WakeupSource of [CanIf\_CheckWakeup()](#_bookmark338) has an invalid value, CanIf shall report development error code CANIF\_E\_PARAM\_WAKE- UPSOURCE to the Det\_ReportError service of the DET, when [CanIf\_Check-](#_bookmark338) [Wakeup()](#_bookmark338) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: The call context of [CanIf\_CheckWakeup()](#_bookmark338) is either on interrupt level (interrupt mode) or on task level (polling mode).

##### CanIf\_CheckValidation [SWS\_CANIF\_00178] *[*

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

Note: *Integration Code* calls this function

**[SWS\_CANIF\_00404]** *[*If parameter WakeupSource of [CanIf\_CheckValidation](#_bookmark341) [()](#_bookmark341) 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()](#_bookmark341) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: The call context of [CanIf\_CheckValidation()](#_bookmark341) 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 corre- sponding mode indications must have been called.

**[SWS\_CANIF\_00408]** *[*Configuration of [CanIf\_CheckValidation()](#_bookmark341): If no valida- tion is needed, this API can be omitted by disabling of [CanIfPublicWakeupCheck-](#_bookmark474) [ValidSupport](#_bookmark474).*]()*

##### CanIf\_GetTxConfirmationState [SWS\_CANIF\_00734] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_GetTxConfirmationState | |
| ***Syntax*** | CanIf\_NotifStatusType CanIf\_GetTxConfirmationState **(**  uint8 ControllerId  **)** | |
| ***Service ID [hex]*** | 0x19 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant (Not for the same controller) | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | [CanIf\_NotifStatusType](#_bookmark253) | Combined TX confirmation status for all TX PDUs of the CAN controller |
| ***Description*** | This service reports, if any TX confirmation has been done for the whole CAN controller since the last CAN controller start. | |
| ***Available via*** | CanIf.h | |

**SWS\_CANIF\_00736]** *[*If parameter ControllerId of [CanIf\_GetTxConfirma-](#_bookmark344) [tionState()](#_bookmark344) 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()](#_bookmark344) is called.*]()*

Note: The call context of [CanIf\_GetTxConfirmationState()](#_bookmark344) is on task level (polling mode).

**[SWS\_CANIF\_00738]** *[*Configuration of [CanIf\_GetTxConfirmationState()](#_bookmark344): If BusOff Recovery of CanSm doesn’t need the status of the Tx confirmations (see [[SWS\_CANIF\_00740](#_bookmark181)]), this API can be omitted by disabling of [CanIfPublic-](#_bookmark472) [TxConfirmPollingSupport](#_bookmark472).*]()*

##### CanIf\_ClearTrcvWufFlag [SWS\_CANIF\_00760] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ClearTrcvWufFlag | |
| ***Syntax*** | Std\_ReturnType CanIf\_ClearTrcvWufFlag **(**  uint8 TransceiverId  **)** | |
| ***Service ID [hex]*** | 0x1e | |
| ***Sync/Async*** | Asynchronous | |
| ***Reentrancy*** | Reentrant for different CAN transceivers | |
| ***Parameters (in)*** | TransceiverId | Abstract CanIf TransceiverId, which is assigned to the designated CAN transceiver. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Request has been accepted E\_NOT\_OK: Request has not been accepted |
| ***Description*** | Requests the CanIf module to clear the WUF flag of the designated CAN transceiver. | |
| ***Available via*** | CanIf.h | |

**[SWS\_CANIF\_00766]** *[*Within [CanIf\_ClearTrcvWufFlag()](#_bookmark346) the function CanTrcv\_ClearTrcvWufFlag() shall be called.*]()*

**[SWS\_CANIF\_00769]** *[*If parameter TransceiverId of [CanIf\_ClearTrcvWuf-](#_bookmark346) [Flag()](#_bookmark346) 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()](#_bookmark346) is caled.*]()*

**[SWS\_CANIF\_00771]** *[*Configuration of [CanIf\_ClearTrcvWufFlag()](#_bookmark346): Whether the CanIf supports this function shall be pre compile time configurable On/Off by the configuration parameter [CanIfPublicPnSupport](#_bookmark466).*]()*

##### CanIf\_CheckTrcvWakeFlag [SWS\_CANIF\_00761] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_CheckTrcvWakeFlag | |
| ***Syntax*** | Std\_ReturnType CanIf\_CheckTrcvWakeFlag **(**  uint8 TransceiverId  **)** | |
| ***Service ID [hex]*** | 0x1f | |
| ***Sync/Async*** | Asynchronous | |
| ***Reentrancy*** | Reentrant for different CAN transceivers | |
| ***Parameters (in)*** | TransceiverId | Abstract CanIf TransceiverId, which is assigned to the designated CAN transceiver. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Request has been accepted E\_NOT\_OK: Request has not been accepted |
| ***Description*** | Requests the CanIf module to check the Wake flag of the designated CAN transceiver. | |
| ***Available via*** | CanIf.h | |

**[SWS\_CANIF\_00765]** *[*Within [CanIf\_CheckTrcvWakeFlag()](#_bookmark348) the function CanTrcv\_CheckWakeFlag() shall be called.*]()*

**[SWS\_CANIF\_00770]** *[*If parameter TransceiverId of [CanIf\_CheckTrcvWake-](#_bookmark348) [Flag()](#_bookmark348) 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()](#_bookmark348) is caled.*]()*

**[SWS\_CANIF\_00813]** *[*Configuration of [CanIf\_CheckTrcvWakeFlag()](#_bookmark348): Whether the CanIf supports this function shall be pre compile time configurable On/Off by the configuration parameter [CanIfPublicPnSupport](#_bookmark466).*]()*

##### CanIf\_SetBaudrate [SWS\_CANIF\_00867] *[*

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

*]()*

**[SWS\_CANIF\_00868]** *[*The service [CanIf\_SetBaudrate()](#_bookmark350) shall call Can\_- SetBaudrate(Controller, BaudRateConfigID) for the requested [CAN Con-](#_bookmark24) [troller](#_bookmark24).*]()*

**[SWS\_CANIF\_00869]** *[*If [CanIf\_SetBaudrate()](#_bookmark350) is called with invalid Control- lerId, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROL- LERID to the Det\_ReportError service of the DET module.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: The parameter BaudRateConfigID of [CanIf\_SetBaudrate()](#_bookmark350) is not checked by [CanIf](#_bookmark8). This has to be done by responsible [CanDrv](#_bookmark5).

Note: The call context of [CanIf\_SetBaudrate()](#_bookmark350) is on task level (polling mode).

**[SWS\_CANIF\_00871]** *[*If [CanIf](#_bookmark8) supports changing baud rate and thus [CanIf\_Set-](#_bookmark350) [Baudrate()](#_bookmark350), shall be configurable via [CanIfSetBaudrateApi](#_bookmark475).*]()*

##### CanIf\_GetControllerRxErrorCounter [SWS\_CANIF\_91003] *[*

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

**[SWS\_CANIF\_00907]** *[*If parameter ControllerId of [CanIf\_GetControllerRx-](#_bookmark353) [ErrorCounter()](#_bookmark353) has an invalid value, the [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROLLERID to the Det\_ReportError service of the DET, when [CanIf\_GetControllerRxErrorCounter()](#_bookmark353) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00908]** *[*If parameter RxErrorCounterPtr of [CanIf\_GetCon-](#_bookmark353) [trollerRxErrorCounter()](#_bookmark353) is a null pointer, the [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_POINTER to the Det\_ReportError service of the DET, when [CanIf\_GetControllerRxErrorCounter()](#_bookmark353) is called.*](*[*SRS\_BSW\_-*](#_bookmark82)[*00323*](#_bookmark82)*)*

##### CanIf\_GetControllerTxErrorCounter [SWS\_CANIF\_91004] *[*

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

**[SWS\_CANIF\_00909]** *[*If parameter ControllerId of [CanIf\_GetControllerTx-](#_bookmark357) [ErrorCounter()](#_bookmark357) has an invalid value, the [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROLLERID to the Det\_ReportError service of the DET, when [CanIf\_GetControllerTxErrorCounter()](#_bookmark357) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00910]** *[*If parameter TxErrorCounterPtr of [CanIf\_GetCon-](#_bookmark357) [trollerTxErrorCounter()](#_bookmark357) is a null pointer, the [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_POINTER to the Det\_ReportError service of the DET, when [CanIf\_GetControllerTxErrorCounter()](#_bookmark357) is called.*](*[*SRS\_BSW\_-*](#_bookmark82)[*00323*](#_bookmark82)*)*

##### CanIf\_EnableBusMirroring [SWS\_CANIF\_91005] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_EnableBusMirroring | |
| ***Syntax*** | Std\_ReturnType CanIf\_EnableBusMirroring **(**  uint8 ControllerId, boolean MirroringActive  **)** | |
| ***Service ID [hex]*** | 0x4c | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller. |
| MirroringActive | TRUE: Mirror\_ReportCanFrame will be called for each frame received or transmitted on the given controller. FALSE: Mirror\_ ReportCanFrame will not be called for the given controller. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | Std\_ReturnType | E\_OK: Mirroring mode was changed.  E\_NOT\_OK: Wrong ControllerId, or mirroring globally disabled (see CanIfBusMirroringSupport). |
| ***Description*** | Enables or disables mirroring for a CAN controller. | |
| ***Available via*** | CanIf.h | |

**[SWS\_CANIF\_00911]** *[*If Bus Mirroring is not enabled (see [CanIfBusMirroring-](#_bookmark462) [Support](#_bookmark462)), the API [CanIf\_EnableBusMirroring()](#_bookmark361) can be omitted.*]([SRS\_Can\_-](#_bookmark138)* [*01172*](#_bookmark138)*)*

**[SWS\_CANIF\_00912]** *[*If parameter ControllerId of [CanIf\_EnableBusMirror-](#_bookmark361) [ing()](#_bookmark361) has an invalid value, the [CanIf](#_bookmark8) shall report development error code CANIF\_- E\_PARAM\_CONTROLLERID to the Det\_ReportError service of the DET, when [CanIf\_EnableBusMirroring()](#_bookmark361) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

##### CanIf\_GetCurrentTime [SWS\_CANIF\_91014]*{*DRAFT*} [*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_GetCurrentTime (draft) | |
| ***Syntax*** | Std\_ReturnType CanIf\_GetCurrentTime **(** uint8 Controller, Can\_TimeStampType**\*** timeStampPtr  **)** | |
| ***Service ID [hex]*** | 0x51 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | Controller | Index of the addresses CAN controller. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | timeStampPtr | current time stamp |
| ***Return value*** | Std\_ReturnType | E\_OK: successful E\_NOT\_OK: failed |
| ***Description*** | This service calls the corresponding CAN Driver service to retrieve the current time value out of the HW registers.  **Tags:** atp.Status=draft | |
| ***Available via*** | CanIf.h | |

*]([SRS\_Can\_01181](#_bookmark139))*

**[SWS\_CANIF\_00922]***{*DRAFT*} [*If development error detection is enabled: the func- tion shall check that the service [CanIf\_Init()](#_bookmark257) 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 func- tion 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 func- tion 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](#_bookmark464)*]()*

##### CanIf\_EnableEgressTimeStamp [SWS\_CANIF\_91011]*{*DRAFT*} [*

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

*]([SRS\_Can\_01181](#_bookmark139))*

**[SWS\_CANIF\_00926]***{*DRAFT*} [*If development error detection is enabled: the func- tion shall check that the service [CanIf\_Init()](#_bookmark257) 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 func- tion 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](#_bookmark464)*]()*

##### CanIf\_GetEgressTimeStamp [SWS\_CANIF\_91012]*{*DRAFT*} [*

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

*]([SRS\_Can\_01181](#_bookmark139))*

**[SWS\_CANIF\_00929]***{*DRAFT*} [*If development error detection is enabled: the func- tion shall check that the service [CanIf\_Init()](#_bookmark257) 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 func- tion 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 func- tion 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](#_bookmark464)*]()*

##### CanIf\_GetIngressTimeStamp [SWS\_CANIF\_91013]*{*DRAFT*} [*

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

*]([SRS\_Can\_01181](#_bookmark139))*

**[SWS\_CANIF\_00933]***{*DRAFT*} [*If development error detection is enabled: the func- tion shall check that the service [CanIf\_Init()](#_bookmark257) 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 func- tion 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 func- tion 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](#_bookmark464)*]()*

#### Callback notifications

This is a list of functions provided for other modules.

##### CanIf\_TriggerTransmit [SWS\_CANIF\_00883] *[*

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

*]()*

**[SWS\_CANIF\_00884]** *[*[CanIf](#_bookmark8) shall only provide the API function [CanIf\_Trigger-](#_bookmark374) [Transmit()](#_bookmark374) if TriggerTransmit support is enabled ([CanIfTriggerTransmitSup-](#_bookmark476) [port](#_bookmark476) = TRUE).*]()*

**[SWS\_CANIF\_00885]** *[*The function [CanIf\_TriggerTransmit()](#_bookmark374) shall call the cor- responding <User\_TriggerTransmit>() function, passing the translated TxPduId and the pointer to the PduInfo structure (PduInfoPtr). Upon return, [CanIf\_Trig-](#_bookmark374) [gerTransmit()](#_bookmark374) shall return the return value of its <User\_TriggerTransmit>().*] ()*

##### CanIf\_TxConfirmation [SWS\_CANIF\_00007] *[*

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

*]([SRS\_Can\_01009](#_bookmark114))*

Note: The service [CanIf\_TxConfirmation()](#_bookmark377) is implemented in [CanIf](#_bookmark8) and called by the [CanDrv](#_bookmark5) after the [CAN L-PDU](#_bookmark3) has been transmitted on the CAN network.

Note: Due to the fact [CanDrv](#_bookmark5) does not support the HandleId concept as described in [[14](#_bookmark49), Specification of ECU Configuration]: Within the service [CanIf\_TxConfirma-](#_bookmark377) [tion()](#_bookmark377), [CanDrv](#_bookmark5) uses PduInfo->swPduHandle as CanTxPduId, which was pre- served from Can\_Write(Hth, \*PduInfo).

**[SWS\_CANIF\_00391]** *[*If configuration parameters [CanIfPublicReadTxPduNoti-](#_bookmark469) [fyStatusApi](#_bookmark469) and [CanIfTxPduReadNotifyStatus](#_bookmark487) for the [Transmitted L-PDU](#_bookmark27) are set to TRUE, and if [CanIf\_TxConfirmation()](#_bookmark377) is called, CanIf shall set the notification status for the [Transmitted L-PDU](#_bookmark27).*]()*

**[SWS\_CANIF\_00410]** *[*If parameter CanTxPduId of [CanIf\_TxConfirmation()](#_bookmark377) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_- LPDU to the Det\_ReportError service of the DET module, when [CanIf\_TxCon-](#_bookmark377) [firmation()](#_bookmark377) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00412]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_TxConfir-](#_bookmark377) [mation()](#_bookmark377), [CanIf](#_bookmark8) shall not call the service <User\_TxConfirmation>() and shall not set the Tx confirmation status, when [CanIf\_TxConfirmation()](#_bookmark377) is called.*]()*

Note: The call context of [CanIf\_TxConfirmation()](#_bookmark377) is either on interrupt level (in- terrupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00414]** *[*Configuration of [CanIf\_TxConfirmation()](#_bookmark377): Each [Tx L-](#_bookmark27)

[-PDU](#_bookmark27) (see [CanIfTxPduCfg](#_bookmark481)) has to be configured with a corresponding transmit con- firmation service of an upper layer module (see [[SWS\_CANIF\_00011](#_bookmark421)]) which is called in [CanIf\_TxConfirmation()](#_bookmark377).*]()*

##### CanIf\_RxIndication [SWS\_CANIF\_00006] *[*

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

*]()*

Note: The service [CanIf\_RxIndication()](#_bookmark383) is implemented in [CanIf](#_bookmark8) and called by [CanDrv](#_bookmark5) after a [CAN L-PDU](#_bookmark3) has been received.

**[SWS\_CANIF\_00415]** *[*Within the service [CanIf\_RxIndication()](#_bookmark383) the [CanIf](#_bookmark8)

routes this indication to the configured upper layer target service(s).*]()*

**[SWS\_CANIF\_00392]** *[*If configuration parameters [CanIfPublicReadRxPduNoti-](#_bookmark468) [fyStatusApi](#_bookmark468) and [CanIfRxPduReadNotifyStatus](#_bookmark503) for the [Received L-PDU](#_bookmark26) are

set to TRUE, and if [CanIf\_RxIndication()](#_bookmark383) is called, the [CanIf](#_bookmark8) shall set the notifi- cation status for the [Received L-PDU](#_bookmark26).*]()*

**[SWS\_CANIF\_00416]** *[*If parameter Mailbox->Hoh of [CanIf\_RxIndication()](#_bookmark383) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_- HOH to the Det\_ReportError service of the DET module, when [CanIf\_RxIndica-](#_bookmark383) [tion()](#_bookmark383) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00417]** *[*If parameter Mailbox->CanId of [CanIf\_RxIndication()](#_bookmark383) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_- CANID to the Det\_ReportError service of the DET module, when [CanIf\_RxIndi-](#_bookmark383) [cation()](#_bookmark383) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

Note: If [CanIf\_RxIndication()](#_bookmark383) is called with invalid PduInfoPtr-> SduLength, runtime error CANIF\_E\_INVALID\_DATA\_LENGTH is reported (see [[SWS\_CANIF\_00168](#_bookmark222)]).

**[SWS\_CANIF\_00419]** *[*If parameter PduInfoPtr or Mailbox of [CanIf\_RxIndi-](#_bookmark383) [cation()](#_bookmark383) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_- E\_PARAM\_POINTER to the Det\_ReportError service of the DET module, when [CanIf\_RxIndication()](#_bookmark383) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00421]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_RxIndica-](#_bookmark383) [tion()](#_bookmark383), [CanIf](#_bookmark8) shall not execute *Rx indication handling*, when [CanIf\_RxIndica-](#_bookmark383) [tion()](#_bookmark383), is called.*]()*

Note: The call context of [CanIf\_RxIndication()](#_bookmark383) is either on interrupt level (inter- rupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00423]** *[*Configuration of [CanIf\_RxIndication()](#_bookmark383): Each [Rx L-PDU](#_bookmark26) (see [CanIfRxPduCfg](#_bookmark495)) has to be configured with a corresponding receive indica- tion service of an upper layer module (see [[SWS\_CANIF\_00012](#_bookmark423)]) which is called in [CanIf\_RxIndication()](#_bookmark383).*]()*

##### CanIf\_ControllerBusOff [SWS\_CANIF\_00218] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ControllerBusOff | |
| ***Syntax*** | void CanIf\_ControllerBusOff **(**  uint8 ControllerId  **)** | |
| ***Service ID [hex]*** | 0x16 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstract CanIf ControllerId which is assigned to a CAN controller, where a BusOff occured. |

|  |  |
| --- | --- |
| ***Parameters (inout)*** | None |
| ***Parameters (out)*** | None |
| ***Return value*** | None |
| ***Description*** | This service indicates a Controller BusOff event referring to the corresponding CAN Controller with the abstract CanIf ControllerId. |
| ***Available via*** | CanIf\_Can.h |

*]()*

Note: The callback service [CanIf\_ControllerBusOff()](#_bookmark389) is called by [CanDrv](#_bookmark5) and implemented in [CanIf](#_bookmark8). It is called in case of a mode change notification of the [CanDrv](#_bookmark5).

**[SWS\_CANIF\_00429]** *[*If parameter ControllerId of [CanIf\_ControllerBusOff](#_bookmark389) [()](#_bookmark389) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_- PARAM\_CONTROLLERID to the Det\_ReportError service of the DET module, when [CanIf\_ControllerBusOff()](#_bookmark389) is called.*](*[*SRS\_BSW\_00323*](#_bookmark82)*)*

**[SWS\_CANIF\_00431]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_Con-](#_bookmark389) [trollerBusOff()](#_bookmark389), [CanIf](#_bookmark8) shall not execute *BusOff notification*, when [CanIf\_Con-](#_bookmark389) [trollerBusOff()](#_bookmark389), is called.*]()*

Note: The call context of [CanIf\_ControllerBusOff()](#_bookmark389) is either on interrupt level (interrupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00433]** *[*Configuration of [CanIf\_ControllerBusOff()](#_bookmark389): ID of the [CAN Controller](#_bookmark24) is published inside the configuration description of the [CanIf](#_bookmark8) (see [CanIfCtrlCfg](#_bookmark525)).*]()*

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

##### CanIf\_ConfirmPnAvailability [SWS\_CANIF\_00815] *[*

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

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

**[SWS\_CANIF\_00753]** *[*If [CanIf\_ConfirmPnAvailability()](#_bookmark392) is called, [CanIf](#_bookmark8)

calls <User\_ConfirmPnAvailability>().*]()*

Note: [CanIf](#_bookmark8) passes the delivered parameter TransceiverId to the upper layer mod- ule.

**[SWS\_CANIF\_00816]** *[*If parameter TransceiverId of [CanIf\_ConfirmPnAvail-](#_bookmark392) [ability()](#_bookmark392) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_- E\_PARAM\_TRCV to the Det\_ReportError service of the DET module, when [CanIf\_ConfirmPnAvailability()](#_bookmark392) is called.*]()*

**[SWS\_CANIF\_00817]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_ConfirmP-](#_bookmark392) [nAvailability()](#_bookmark392), [CanIf](#_bookmark8) shall not execute notification, when [CanIf\_ConfirmP-](#_bookmark392) [nAvailability()](#_bookmark392) is called.*]()*

Note: The call context of [CanIf\_ConfirmPnAvailability()](#_bookmark392) is either on interrupt level (interrupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00754]** *[*Configuration of [CanIf\_ConfirmPnAvailability()](#_bookmark392): This function shall be pre compile time configurable ON/OFF by the configuration parameter [CanIfPublicPnSupport](#_bookmark466).*]()*

##### CanIf\_ClearTrcvWufFlagIndication [SWS\_CANIF\_00762] *[*

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

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

**[SWS\_CANIF\_00757]** *[*If [CanIf\_ClearTrcvWufFlagIndication()](#_bookmark395) is called,

[CanIf](#_bookmark8) calls <User\_ClearTrcvWufFlagIndication>().*]()*

Note: [CanIf](#_bookmark8) passes the delivered parameter TransceiverId to the upper layer mod- ule.

**[SWS\_CANIF\_00805]** *[*If parameter TransceiverId of [CanIf\_ClearTrcvWuf-](#_bookmark395) [FlagIndication()](#_bookmark395) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_TRCV to the Det\_ReportError service of the DET module, when [CanIf\_ClearTrcvWufFlagIndication()](#_bookmark395) is called.*]()*

**[SWS\_CANIF\_00806]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_ClearTr-](#_bookmark395) [cvWufFlagIndication()](#_bookmark395), [CanIf](#_bookmark8) shall not execute notification, when [CanIf\_-](#_bookmark395) [ClearTrcvWufFlagIndication()](#_bookmark395) is called.*]()*

Note: The call context of [CanIf\_ClearTrcvWufFlagIndication()](#_bookmark395) is either on interrupt level (interrupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00808]** *[*Configuration of [CanIf\_ClearTrcvWufFlagIndication](#_bookmark395) [()](#_bookmark395): This function shall be pre compile time configurable ON/OFF by the configuration parameter [CanIfPublicPnSupport](#_bookmark466).*]()*

##### CanIf\_CheckTrcvWakeFlagIndication [SWS\_CANIF\_00763] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_CheckTrcvWakeFlagIndication | |
| ***Syntax*** | void CanIf\_CheckTrcvWakeFlagIndication **(**  uint8 TransceiverId  **)** | |
| ***Service ID [hex]*** | 0x21 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstract CanIf TransceiverId, which is assigned to a CAN transceiver, for which this function was called. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | This service indicates that the check of the transceiver’s wake-up flag has been finished by the corresponding CAN transceiver with the abstract CanIf TransceiverId. This indication is used to cope with the asynchronous transceiver communication. | |
| ***Available via*** | CanIf\_CanTrcv.h | |

**[SWS\_CANIF\_00759]** *[*If [CanIf\_CheckTrcvWakeFlagIndication()](#_bookmark398) is called,

[CanIf](#_bookmark8) calls <User\_CheckTrcvWakeFlagIndication>().*]()*

Note: [CanIf](#_bookmark8) passes the delivered parameter TransceiverId to the upper layer mod- ule.

**[SWS\_CANIF\_00809]** *[*If parameter TransceiverId of [CanIf\_CheckTrcvWake-](#_bookmark398) [FlagIndication()](#_bookmark398) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_TRCV to the Det\_ReportError service of the DET module, when [CanIf\_CheckTrcvWakeFlagIndication()](#_bookmark398) is called.*]()*

**[SWS\_CANIF\_00810]** *[*If the CanIf was not initialized before calling [CanIf\_Check-](#_bookmark398) [TrcvWakeFlagIndication()](#_bookmark398), [CanIf](#_bookmark8) shall not execute notification, when [CanIf\_-](#_bookmark398) [CheckTrcvWakeFlagIndication()](#_bookmark398) is called.*]()*

Note: The call context of [CanIf\_CheckTrcvWakeFlagIndication()](#_bookmark398) is either on interrupt level (interrupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00812]** *[*Configuration of [CanIf\_CheckTrcvWakeFlagIndication](#_bookmark398) [()](#_bookmark398): This function shall be pre compile time configurable ON/OFF by the configuration parameter [CanIfPublicPnSupport](#_bookmark466).*]()*

##### CanIf\_ControllerModeIndication [SWS\_CANIF\_00699] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ControllerModeIndication | |
| ***Syntax*** | void CanIf\_ControllerModeIndication **(** uint8 ControllerId, Can\_ControllerStateType ControllerMode  **)** | |
| ***Service ID [hex]*** | 0x17 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstract CanIf ControllerId which is assigned to a CAN controller, which state has been transitioned. |
| ControllerMode | Mode to which the CAN controller transitioned |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | This service indicates a controller state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId. | |
| ***Available via*** | CanIf\_Can.h | |

Note: The callback service [CanIf\_ControllerModeIndication()](#_bookmark402) is called by [CanDrv](#_bookmark5) and implemented in [CanIf](#_bookmark8). It is called in case of a state transition notification of the [CanDrv](#_bookmark5).

**[SWS\_CANIF\_00700]** *[*If parameter ControllerId of [CanIf\_ControllerMod-](#_bookmark402) [eIndication()](#_bookmark402) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROLLERID to the Det\_ReportError service of the DET module, when [CanIf\_ControllerModeIndication()](#_bookmark402) is called.*]()*

**[SWS\_CANIF\_00702]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_Con-](#_bookmark402) [trollerModeIndication()](#_bookmark402), [CanIf](#_bookmark8) shall not execute state transition notification, when [CanIf\_ControllerModeIndication()](#_bookmark402) is called.*]()*

Note: The call context of [CanIf\_ControllerModeIndication()](#_bookmark402) is either on inter- rupt level (interrupt mode) or on task level (polling mode).

##### CanIf\_TrcvModeIndication [SWS\_CANIF\_00764] *[*

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

*]()*

Note: The callback service [CanIf\_TrcvModeIndication()](#_bookmark404) is called by [CanDrv](#_bookmark5) and implemented in [CanIf](#_bookmark8). It is called in case of a state transition notification of the [CanDrv](#_bookmark5).

**[SWS\_CANIF\_00706]** *[*If parameter TransceiverId of [CanIf\_TrcvModeIndica-](#_bookmark404) [tion()](#_bookmark404) has an invalid value, [CanIf](#_bookmark8) shall report development error code CANIF\_E\_- PARAM\_TRCV to the Det\_ReportError service of the DET module, when [CanIf\_-](#_bookmark404) [TrcvModeIndication()](#_bookmark404) is called.*]()*

**[SWS\_CANIF\_00708]** *[*If [CanIf](#_bookmark8) was not initialized before calling [CanIf\_TrcvMod-](#_bookmark404) [eIndication()](#_bookmark404), [CanIf](#_bookmark8) shall not execute state transition notification, when [CanIf\_-](#_bookmark404) [TrcvModeIndication()](#_bookmark404) is called.*]()*

Note: The call context of [CanIf\_TrcvModeIndication()](#_bookmark404) is either on interrupt level (interrupt mode) or on task level (polling mode).

**[SWS\_CANIF\_00710]** *[*Configuration of [CanIf\_TrcvModeIndication()](#_bookmark404): ID of the [CAN Transceiver](#_bookmark12) is published inside the configuration description of [CanIf](#_bookmark8) via pa- rameter [CanIfTrcvId](#_bookmark535).*]()*

**[SWS\_CANIF\_00730]** *[*Configuration of [CanIf\_TrcvModeIndication()](#_bookmark404): If transceivers are not supported (CanIfTrcvDrvCfg is not configured, see [CanIfTr-](#_bookmark532) [cvDrvCfg](#_bookmark532)), [CanIf\_TrcvModeIndication()](#_bookmark404) shall not be provided by [CanIf](#_bookmark8).*]()*

##### CanIf\_ControllerErrorStatePassive [SWS\_CANIF\_91008] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ControllerErrorStatePassive | |
| ***Syntax*** | void CanIf\_ControllerErrorStatePassive **(**  uint8 ControllerId, uint16 RxErrorCounter, uint16 TxErrorCounter  **)** | |
| ***Service ID [hex]*** | 0x4f | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller. |
| RxErrorCounter | Value of the Rx error counter |
| TxErrorCounter | Value of the Tx error counter |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | void | – |
| ***Description*** | The function derives the ErrorCounterTreshold from RxErrorCounter/ TxErrorCounter values and reports it to the IdsM as security event CANIF\_SEV\_ERRORSTATE\_PASSIVE to the IdsM. It also prepares the context data for the respective security event. | |
| ***Available via*** | CanIf\_Can.h | |

*]([RS\_Ids\_00810](#_bookmark67))*

**[SWS\_CANIF\_00919]** *[*If parameter ControllerId of [CanIf\_Con-](#_bookmark407) [trollerErrorStatePassive()](#_bookmark407) has an invalid value, the [CanIf](#_bookmark8) shall report development error code CANIF\_E\_PARAM\_CONTROLLERID to the Det\_ReportEr- ror service of the DET module when [CanIf\_ControllerErrorStatePassive()](#_bookmark407) is called.*](*[*RS\_Ids\_00810*](#_bookmark67)*)*

##### CanIf\_ErrorNotification [SWS\_CANIF\_91009] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | CanIf\_ErrorNotification | |
| ***Syntax*** | void CanIf\_ErrorNotification **(** uint8 ControllerId, Can\_ErrorType Can\_ErrorType  **)** | |
| ***Service ID [hex]*** | 0x50 | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller. |
| Can\_ErrorType | Reported CAN error |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | void | – |
| ***Description*** | The function shall derive the bus error source rx or tx from the parameter CanError and report the bus error as security event CANIF\_SEV\_TX\_ERROR\_DETECTED or CANIF\_SEV\_RX\_ ERROR\_DETECTED. It also prepares the context data for the respective security event. | |
| ***Available via*** | CanIf\_Can.h | |

*]([RS\_Ids\_00810](#_bookmark67))*

**[SWS\_CANIF\_00920]** *[*If parameter ControllerId of [CanIf\_ErrorNotifica-](#_bookmark411) [tion()](#_bookmark411) has an invalid value, the [CanIf](#_bookmark8) shall report development error code CANIF\_- E\_PARAM\_CONTROLLERID to the Det\_ReportError service of the DET module, when [CanIf\_ErrorNotification()](#_bookmark411) is called.*](*[*RS\_Ids\_00810*](#_bookmark67)*)*

**[SWS\_CANIF\_00921]** *[*If parameter CanError of [CanIf\_ErrorNotification()](#_bookmark411) has an invalid value, the [CanIf](#_bookmark8) shall report development error code CANIF\_E\_- PARAM\_CAN\_ERROR to the Det\_ReportError service of the DET module, when [CanIf\_ErrorNotification()](#_bookmark411) is called.*](*[*RS\_Ids\_00810*](#_bookmark67)*)*

#### Scheduled functions

Note: [CanIf](#_bookmark8) does not have scheduled functions or needs some.

#### Expected interfaces

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

##### Mandatory interfaces

Note: This section defines all interfaces, which are required to fulfill the core function- ality 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. |

##### 

##### 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. |
| 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. |
| 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. |
| 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. |

##### 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](#_bookmark8) according to the AUTOSAR BSW architecture. The specific callback notification is specified in the corresponding SWS document (see [chapter](#_bookmark34) [3](#_bookmark34) “[Related documentation](#_bookmark34)”).

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](#_bookmark8) 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](#_bookmark8). This condition displays the symbolic name of the functional group in a callback service in the corre- sponding 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](#_bookmark4) ID.

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

###### <User\_TriggerTransmit> [SWS\_CANIF\_00886] *[*

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

Note: This callback service is called by [CanIf](#_bookmark8) and implemented in the corresponding upper layer module. It is called in case of a *Trigger Transmit* request of [CanDrv](#_bookmark5).

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 con- figured by [CanIfTxPduUserTxConfirmationUL](#_bookmark492) (see [CanIfTxPduUserTxCon-](#_bookmark492) [firmationUL](#_bookmark492)). If no upper layer modules are configured, no TriggerTransmit call- back 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](#_bookmark8) shall be con- figured for [CanIf](#_bookmark8) by parameter [CanIfTxPduUserTriggerTransmitName](#_bookmark490) (see [CanIfTxPduUserTriggerTransmitName](#_bookmark490)).*]()*

Note: If [CanIfTxPduTriggerTransmit](#_bookmark488) is not specified or FALSE, no upper layer modules have to be configured for *Trigger Transmit*. Therefore, <User\_Trigger- Transmit>() will not be called and [CanIfTxPduUserTxConfirmationUL](#_bookmark492) as well as [CanIfTxPduUserTriggerTransmitName](#_bookmark490) need not to be configured.

**[SWS\_CANIF\_00890]** *[*Configuration of <User\_TriggerTransmit>(): If [CanI-](#_bookmark492) [fTxPduUserTxConfirmationUL](#_bookmark492) is set to PDUR, [CanIfTxPduUserTrigger-](#_bookmark490) [TransmitName](#_bookmark490) must be PduR\_CanIfTriggerTransmit.*]()*

**[SWS\_CANIF\_00891]** *[*Configuration of <User\_TriggerTransmit>(): If [CanI-](#_bookmark492) [fTxPduUserTxConfirmationUL](#_bookmark492) is set to CDD, the name of the API <User\_Trig- gerTransmit>() has to be configured via parameter [CanIfTxPduUserTrigger-](#_bookmark490) [TransmitName](#_bookmark490).*]()*

###### <User\_TxConfirmation> [SWS\_CANIF\_00011] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | <User\_TxConfirmation> | |
| ***Syntax*** | void <User\_TxConfirmation> **(** PduIdType TxPduId, Std\_ReturnType result  **)** | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant for different PduIds. Non reentrant for the same PduId. | |
| ***Parameters (in)*** | TxPduId | ID of the PDU that has been transmitted. |
| result | E\_OK: The PDU was transmitted. E\_NOT\_OK: Transmission of the PDU failed. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU. | |
| ***Available via*** | configurable | |

Note: This callback service is called by [CanIf](#_bookmark8) and implemented in the corresponding upper layer module. It is called in case of a *transmit confirmation* of [CanDrv](#_bookmark5).

Note: This type of confirmation callback service is mainly designed for [PduR](#_bookmark22), [CanNm](#_bookmark9), and [CanTp](#_bookmark11), 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 [CanI-](#_bookmark492) [fTxPduUserTxConfirmationUL](#_bookmark492). If no upper layer modules are configured for *trans- mit confirmation* using <User\_TxConfirmation>(), no *transmit confirmation* is ex- ecuted.*]()*

**[SWS\_CANIF\_00542]** *[*Configuration of <User\_TxConfirmation>(): The name of the API <User\_TxConfirmation>() which is called by [CanIf](#_bookmark8) shall be configured for [CanIf](#_bookmark8) by parameter [CanIfTxPduUserTxConfirmationName](#_bookmark491).*]()*

Note: If *transmit confirmations* are not necessary or no upper layer modules are con- figured for *transmit confirmations* and thus <User\_TxConfirmation>() shall not be called, [CanIfTxPduUserTxConfirmationUL](#_bookmark492) and [CanIfTxPduUserTxConfir-](#_bookmark491) [mationName](#_bookmark491) need not to be configured.

**[SWS\_CANIF\_00439]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to PDUR, [CanIfTxPduUserTxConfirmation-](#_bookmark491) [Name](#_bookmark491) must be PduR\_CanIfTxConfirmation.*]()*

**[SWS\_CANIF\_00543]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to CAN\_NM, [CanIfTxPduUserTxConfirma-](#_bookmark491) [tionName](#_bookmark491) must be CanNm\_TxConfirmation.*]()*

Hint (Dependency to another module):

If at least one [CanIf](#_bookmark8) Tx [L-SDU](#_bookmark4) is configured with CanNm\_TxConfirmation(), which means [CanIfTxPduUserTxConfirmationUL](#_bookmark492) equals CAN\_NM, the [CanNm](#_bookmark9) config- uration parameter CANNM\_IMMEDIATE\_TXCONF\_ENABLED must be set to FALSE (for [CanNm](#_bookmark9) related details see [[4](#_bookmark39), Specification of CAN Network Management], [SWS\_CANNM\_00284]).

**[SWS\_CANIF\_00858]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to J1939NM, [CanIfTxPduUserTxConfirma-](#_bookmark491) [tionName](#_bookmark491) must be J1939Nm\_TxConfirmation.*]()*

**[SWS\_CANIF\_00544]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to J1939TP, [CanIfTxPduUserTxConfirma-](#_bookmark491) [tionName](#_bookmark491) must be J1939Tp\_TxConfirmation.*]()*

**[SWS\_CANIF\_00550]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to CAN\_TP, [CanIfTxPduUserTxConfirma-](#_bookmark491) [tionName](#_bookmark491) must be CanTp\_TxConfirmation.*]()*

**[SWS\_CANIF\_00556]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to XCP, [CanIfTxPduUserTxConfirmation-](#_bookmark491) [Name](#_bookmark491) must be Xcp\_CanIfTxConfirmation.*]()*

**[SWS\_CANIF\_00551]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to CDD, the name of the API <User\_TxCon- firmation>() has to be configured via parameter [CanIfTxPduUserTxConfir-](#_bookmark491) [mationName](#_bookmark491).*]()*

**[SWS\_CANIF\_00879]** *[*Configuration of <User\_TxConfirmation>(): If [CanIfTx-](#_bookmark492) [PduUserTxConfirmationUL](#_bookmark492) is set to CAN\_TSYN, [CanIfTxPduUserTxConfirma-](#_bookmark491) [tionName](#_bookmark491) must be CanTSyn\_TxConfirmation.*]()*

###### <User\_RxIndication> [SWS\_CANIF\_00012] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | <User\_RxIndication> | |
| ***Syntax*** | void <User\_RxIndication> **(**  PduIdType RxPduId,  const PduInfoType**\*** PduInfoPtr  **)** | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Reentrant for different PduIds. Non reentrant for the same PduId. | |
| ***Parameters (in)*** | 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. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | Indication of a received PDU from a lower layer communication interface module. | |
| ***Available via*** | configurable | |

*]([SRS\_Can\_01003](#_bookmark111))*

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](#_bookmark8) and implemented in the configured upper layer module (e.g. [PduR](#_bookmark22), [CanNm](#_bookmark9), [CanTp](#_bookmark11), etc.) if configured accordingly (see [CanIfRxPduUserRxIndicationUL](#_bookmark505)).

Note: Until <User\_RxIndication>() returns, [CanIf](#_bookmark8) will not access <PduIn- foPtr>. The <PduInfoPtr> is only valid and can be used by upper layers, until the indication returns. [CanIf](#_bookmark8) guarantees that the number of configured bytes for this

<PduInfoPtr> is valid.

Note: The call context of <User\_RxIndication>() is either on interrupt level (*inter- rupt 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 [CanIfRxPdu-](#_bookmark505) [UserRxIndicationUL](#_bookmark505).*]()*

**[SWS\_CANIF\_00552]** *[*Configuration of <User\_RxIndication>(): The name of the API <User\_RxIndication>() which will be called by [CanIf](#_bookmark8) shall be configured for [CanIf](#_bookmark8) by parameter [CanIfRxPduUserRxIndicationName](#_bookmark504).*]()*

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, [Can-](#_bookmark505) [IfRxPduUserRxIndicationUL](#_bookmark505) and [CanIfRxPduUserRxIndicationName](#_bookmark504) need not to be configured.

**[SWS\_CANIF\_00442]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to PDUR, [CanIfRxPduUserRxIndicationName](#_bookmark504) must be PduR\_CanIfRxIndication.*]()*

**[SWS\_CANIF\_00445]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to CAN\_NM, [CanIfRxPduUserRxIndicationName](#_bookmark504) 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](#_bookmark9) related details see [[4](#_bookmark39), Specifica- tion of CAN Network Management]).

**[SWS\_CANIF\_00859]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to J1939NM, [CanIfRxPduUserRxIndication-](#_bookmark504) [Name](#_bookmark504) must be J1939Nm\_RxIndication.*]()*

**[SWS\_CANIF\_00448]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to CAN\_TP, [CanIfRxPduUserRxIndicationName](#_bookmark504) must be CanTp\_RxIndication.*]()*

**[SWS\_CANIF\_00554]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to J1939TP, [CanIfRxPduUserRxIndication-](#_bookmark504) [Name](#_bookmark504) must be J1939Tp\_RxIndication.*]()*

**[SWS\_CANIF\_00555]** *[*Configuration of <User\_RxIndication>(): If [CanIfRx-](#_bookmark505) [PduUserRxIndicationUL](#_bookmark505) is set to XCP, [CanIfRxPduUserRxIndicationName](#_bookmark504) must be Xcp\_CanIfRxIndication.*]()*

**[SWS\_CANIF\_00557]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to CDD the name of the API has to be configured via parameter [CanIfRxPduUserRxIndicationName](#_bookmark504).*]()*

**[SWS\_CANIF\_00880]** *[*Configuration of <User\_RxIndication>(): If [CanIfRxP-](#_bookmark505) [duUserRxIndicationUL](#_bookmark505) is set to CAN\_TSYN, [CanIfRxPduUserRxIndication-](#_bookmark504) [Name](#_bookmark504) must be CanTSyn\_RxIndication.*]()*

###### <User\_ValidateWakeupEvent> [SWS\_CANIF\_00532] *[*

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

Note: This callback service is mainly implemented in and used by the *ECU State Man- ager* module (see [[13](#_bookmark48), Specification of ECU State Manager]).

Note: The [CanIf](#_bookmark8) calls this callback service. It is implemented by the configured up- per layer module. It is called only during the call of [CanIf\_CheckValidation()](#_bookmark341) if a first CAN L-PDU reception event after a wake up event has been occurred at the corresponding [CAN Controller](#_bookmark24).

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 [CanIfPublicWake-](#_bookmark474) [upCheckValidSupport](#_bookmark474).*]()*

**[SWS\_CANIF\_00456]** *[*Configuration of <User\_ValidateWakeupEvent>(): The upper layer module which provides this callback service has to be configured by [Can-](#_bookmark523) [IfDispatchUserValidateWakeupEventUL](#_bookmark523), but:

* If no upper layer modules are configured for wake up notification using <User\_- ValidateWakeupEvent>(), no wake up notification needs to be configured. [CanIfDispatchUserValidateWakeupEventUL](#_bookmark523) needs not to be configured.
* If wake up is not supported ([CanIfCtrlWakeupSupport](#_bookmark527) and [CanIfTr-](#_bookmark536) [cvWakeupSupport](#_bookmark536) equal FALSE, [CanIfDispatchUserValidateWakeu-](#_bookmark523) [pEventUL](#_bookmark523) is not configurable.

**[SWS\_CANIF\_00563]** *[*Configuration of <User\_ValidateWakeupEvent>(): If [CanIfDispatchUserValidateWakeupEventUL](#_bookmark523) is set to ECUM, [CanIfDis-](#_bookmark522) [patchUserValidateWakeupEventName](#_bookmark522) must be EcuM\_ValidateWakeu- pEvent.*]()*

**[SWS\_CANIF\_00564]** *[*Configuration of <User\_ValidateWakeupEvent>(): If [CanIfDispatchUserValidateWakeupEventUL](#_bookmark523) is set to CDD the name of the API has to be configured via parameter [CanIfDispatchUserValidateWakeupEvent-](#_bookmark522) [Name](#_bookmark522).*]()*

###### <User\_ControllerBusOff> [SWS\_CANIF\_00014] *[*

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

*]([SRS\_Can\_01029](#_bookmark123))*

Note: This callback service is mainly implemented in and used by [CanSm](#_bookmark10) (see [[3](#_bookmark38), Specification of CAN State Manager]).

Note: This callback service is called by [CanIf](#_bookmark8) and implemented by the configured up- per layer module. It is called in case of a *BusOff notification* via [CanIf\_Controller-](#_bookmark389) [BusOff()](#_bookmark389) of the [CanDrv](#_bookmark5). The delivered parameter ControllerId of the service [CanIf\_ControllerBusOff()](#_bookmark389) 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](#_bookmark24) usage, but not for the same [CAN Controller](#_bookmark24).

Note: Before re-initialization/restart during *BusOff recovery* is executed <User\_Con- trollerBusOff>() is performed only once in case of multiple *BusOff events* at [CAN](#_bookmark24) [Controller](#_bookmark24).

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 [CanIfDis-](#_bookmark517) [patchUserCtrlBusOffUL](#_bookmark517).*]()*

**[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](#_bookmark516).*]()*

**[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()](#_bookmark389).*]()*

**[SWS\_CANIF\_00559]** *[*Configuration of <User\_ControllerBusOff>(): If [Can-](#_bookmark517) [IfDispatchUserCtrlBusOffUL](#_bookmark517) is set to CAN\_SM, [CanIfDispatchUserCtrlBu-](#_bookmark516) [sOffName](#_bookmark516) must be CanSM\_ControllerBusOff.*]()*

**[SWS\_CANIF\_00560]** *[*Configuration of <User\_ControllerBusOff>(): If [Can-](#_bookmark517) [IfDispatchUserCtrlBusOffUL](#_bookmark517) is set to CDD the name of the API has to be config- ured via parameter [CanIfDispatchUserCtrlBusOffName](#_bookmark516).*]()*

###### <User\_ConfirmPnAvailability> [SWS\_CANIF\_00821] *[*

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

Note: This callback service is mainly implemented in and used by [CanSm](#_bookmark10) (see [[3](#_bookmark38), 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](#_bookmark393)]), has to be config- urable by [CanIfDispatchUserConfirmPnAvailabilityUL](#_bookmark515) if [CanIfPublicPn-](#_bookmark466) [Support](#_bookmark466) equals True.*]()*

**[SWS\_CANIF\_00824]** *[*Configuration of <User\_ConfirmPnAvailability>()

: The name of <User\_ConfirmPnAvailability>() shall be configurable by [CanIfDispatchUserConfirmPnAvailabilityName](#_bookmark514) if [CanIfPublicPnSup-](#_bookmark466) [port](#_bookmark466) equals True.*]()*

**[SWS\_CANIF\_00825]** *[*Configuration of <User\_ConfirmPnAvailability>(): It shall be configurable by [CanIfPublicPnSupport](#_bookmark466), if [CanIf](#_bookmark8) supports this service ( False: not supported, True: supported)*]()*

**[SWS\_CANIF\_00826]** *[*Configuration of <User\_ConfirmPnAvailability>(): If [CanIfDispatchUserConfirmPnAvailabilityUL](#_bookmark515) is set to CAN\_SM, [CanIfDis-](#_bookmark514) [patchUserConfirmPnAvailabilityName](#_bookmark514) must be CanSM\_ConfirmPnAvail- ability.*]()*

**[SWS\_CANIF\_00827]** *[*Configuration of <User\_ConfirmPnAvailability>(): If [CanIfDispatchUserConfirmPnAvailabilityUL](#_bookmark515) is set to CDD, the name of the service has to be configurable via parameter [CanIfDispatchUserConfirmP-](#_bookmark514) [nAvailabilityName](#_bookmark514).*]()*

###### <User\_ClearTrcvWufFlagIndication> [SWS\_CANIF\_00788] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | <User\_ClearTrcvWufFlagIndication> | |
| ***Syntax*** | void <User\_ClearTrcvWufFlagIndication> **(**  uint8 TransceiverId  **)** | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstracted CanIf TransceiverId, for which this function was called. |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | This service indicates that the CAN transceiver has cleared the WufFlag. This function is called in CanIf\_ClearTrcvWufFlagIndication. | |
| ***Available via*** | configurable | |

Note: This callback service is mainly implemented in and used by [CanSm](#_bookmark10) (see [[3](#_bookmark38), 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 gen- eral 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](#_bookmark396)]), has to be configurable by [CanIfDispatchUserClearTrcvWufFlagIndicationUL](#_bookmark513) if [Can-](#_bookmark466) [IfPublicPnSupport](#_bookmark466) equals True.*]()*

**[SWS\_CANIF\_00795]** *[*Configuration of <User\_ClearTrcvWufFlagIndication> (): The name of <User\_ClearTrcvWufFlagIndication>() shall be config- urable by [CanIfDispatchUserClearTrcvWufFlagIndicationName](#_bookmark512) if [CanIf-](#_bookmark466) [PublicPnSupport](#_bookmark466) equals True.*]()*

**[SWS\_CANIF\_00796]** *[*Configuration of <User\_ClearTrcvWufFlagIndication> (): It shall be configurable by [CanIfPublicPnSupport](#_bookmark466), if [CanIf](#_bookmark8) supports this ser- vice (False: not supported, True: supported)*]()*

**[SWS\_CANIF\_00797]** *[*Configuration of <User\_ClearTrcvWufFlagIndication> (): If [CanIfDispatchUserClearTrcvWufFlagIndicationUL](#_bookmark513) is set to CAN\_SM, [CanIfDispatchUserClearTrcvWufFlagIndicationName](#_bookmark512) must be CanSM\_- ClearTrcvWufFlagIndication.*]()*

**[SWS\_CANIF\_00798]** *[*Configuration of <User\_ClearTrcvWufFlagIndication> (): If [CanIfDispatchUserClearTrcvWufFlagIndicationUL](#_bookmark513) is set to CDD, the name of the service has to be configurable via parameter [CanIfDispatchUser-](#_bookmark512) [ClearTrcvWufFlagIndicationName](#_bookmark512).*]()*

###### <User\_CheckTrcvWakeFlagIndication> [SWS\_CANIF\_00814] *[*

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

Note: This callback service is mainly implemented in and used by [CanSm](#_bookmark10) (see [[3](#_bookmark38), 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 gen- eral re-entrant for multiple CAN Controller usage, but not for the same CAN Controller

**[SWS\_CANIF\_00800]** *[*Configuration of <User\_CheckTrcvWakeFlagIndica- tion>(): The upper layer module, which is called (see [[SWS\_CANIF\_00759](#_bookmark399)]), has to be configurable by [CanIfDispatchUserCheckTrcvWakeFlagIndicationUL](#_bookmark511) if [CanIfPublicPnSupport](#_bookmark466) equals True.*]()*

**[SWS\_CANIF\_00801]** *[*Configuration of <User\_CheckTrcvWakeFlagIndica- tion>(): The name of <User\_CheckTrcvWakeFlagIndication>() shall be con- figurable by [CanIfDispatchUserCheckTrcvWakeFlagIndicationName](#_bookmark510) if [Can-](#_bookmark466) [IfPublicPnSupport](#_bookmark466) equals True.*]()*

**[SWS\_CANIF\_00802]** *[*Configuration of <User\_CheckTrcvWakeFlagIndica- tion>(): It shall be configurable by [CanIfPublicPnSupport](#_bookmark466), if [CanIf](#_bookmark8) supports this service (False: not supported, True: supported)*]()*

**[SWS\_CANIF\_00803]** *[*Configuration of <User\_CheckTrcvWakeFlagIndica- tion>(): If [CanIfDispatchUserCheckTrcvWakeFlagIndicationUL](#_bookmark511) is set to CAN\_SM, [CanIfDispatchUserCheckTrcvWakeFlagIndicationName](#_bookmark510) must be CanSM\_CheckTransceiverWakeFlagIndication.*]()*

**[SWS\_CANIF\_00804]** *[*Configuration of <User\_CheckTrcvWakeFlagIndica- tion>(): If [CanIfDispatchUserCheckTrcvWakeFlagIndicationUL](#_bookmark511) is set to CDD, the name of the service has to be configurable via parameter [CanIfDis-](#_bookmark510) [patchUserCheckTrcvWakeFlagIndicationName](#_bookmark510).*]()*

###### <User\_ControllerModeIndication> [SWS\_CANIF\_00687] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | <User\_ControllerModeIndication> | |
| ***Syntax*** | void <User\_ControllerModeIndication> **(** uint8 ControllerId, Can\_ControllerStateType ControllerMode  **)** | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller, at which a controller state transition occurred. |
| ControllerMode | Notified CAN controller mode |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | This service indicates a CAN controller state transition to the corresponding upper layer module (mainly the CAN State Manager module). | |
| ***Available via*** | configurable | |

Note: The upper layer module provides the Service ID.

Note: This callback service is mainly implemented in and used by [CanSm](#_bookmark10) (see [[3](#_bookmark38), Specification of CAN State Manager]).

Note: The [CanIf](#_bookmark8) calls this callback service. It is implemented by the configured up- per layer module. It is called in case of a *state transition notification* via [CanIf\_-](#_bookmark402) [ControllerModeIndication()](#_bookmark402) of the [CanDrv](#_bookmark5). The delivered parameter Con- trollerId of the service [CanIf\_ControllerModeIndication()](#_bookmark402) is passed to the upper layer module. The delivered parameter ControllerMode of the service [CanIf\_ControllerModeIndication()](#_bookmark402) 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](#_bookmark519).*]()*

**[SWS\_CANIF\_00690]** *[*Configuration of <User\_ControllerModeIndication>()

: The name of <User\_ControllerModeIndication>() which is called by [CanIf](#_bookmark8) shall be configured for [CanIf](#_bookmark8) by parameter [CanIfDispatchUserCtrlModeIndi-](#_bookmark518) [cationName](#_bookmark518). This is only necessary if *state transition notifications* are configured via [CanIfDispatchUserCtrlModeIndicationUL](#_bookmark519).*]()*

**[SWS\_CANIF\_00691]** *[*Configuration of <User\_ControllerModeIndication>()

: If [CanIfDispatchUserCtrlModeIndicationUL](#_bookmark519) is set to CAN\_SM, [CanIfDis-](#_bookmark518) [patchUserCtrlModeIndicationName](#_bookmark518) must be CanSM\_ControllerModeIndi- cation.*]()*

**[SWS\_CANIF\_00692]** *[*Configuration of <User\_ControllerModeIndication>()

: If [CanIfDispatchUserCtrlModeIndicationUL](#_bookmark519) is set to CDD the name of the function has to be configured via parameter [CanIfDispatchUserCtrlModeIndi-](#_bookmark518) [cationName](#_bookmark518).*]()*

###### <User\_TrcvModeIndication> [SWS\_CANIF\_00693] *[*

|  |  |  |
| --- | --- | --- |
| ***Service Name*** | <User\_TrcvModeIndication> | |
| ***Syntax*** | void <User\_TrcvModeIndication> **(** uint8 TransceiverId, CanTrcv\_TrcvModeType TransceiverMode  **)** | |
| ***Sync/Async*** | Synchronous | |
| ***Reentrancy*** | Non Reentrant | |
| ***Parameters (in)*** | TransceiverId | Abstracted CanIf TransceiverId which is assigned to a CAN transceiver, at which a transceiver state transition occurred. |
| TransceiverMode | Notified CAN transceiver mode |
| ***Parameters (inout)*** | None | |
| ***Parameters (out)*** | None | |
| ***Return value*** | None | |
| ***Description*** | This service indicates a CAN transceiver state transition to the corresponding upper layer module (mainly the CAN State Manager module). | |
| ***Available via*** | configurable | |

Note: The upper layer module provides the Service ID.

Note: This callback service is mainly implemented in and used by [CanSm](#_bookmark10) (see [[3](#_bookmark38), Specification of CAN State Manager]).

Note: The [CanIf](#_bookmark8) 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\_TrcvMod-](#_bookmark404) [eIndication()](#_bookmark404) of the [CanTrcv](#_bookmark12). The delivered parameter Transceiver of the ser- vice [CanIf\_TrcvModeIndication()](#_bookmark404) is mapped (as configured) to the appropriate parameter TransceiverId which will be passed to the upper layer module. The de- livered parameter TransceiverMode of the service [CanIf\_TrcvModeIndication](#_bookmark404) [()](#_bookmark404) is mapped to the appropriate parameter TransceiverMode of <User\_TrcvMod- eIndication>().

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

**[SWS\_CANIF\_00694]** *[*Caveats of <User\_TrcvModeIndication>():

* The [CanTrcv](#_bookmark12) 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](#_bookmark12) us- age, but not for the same [CAN Transceiver](#_bookmark12).

**[SWS\_CANIF\_00695]** *[*Configuration of <User\_TrcvModeIndication>(): The up- per layer module which provides this callback service has to be configured by [Can-](#_bookmark521) [IfDispatchUserTrcvModeIndicationUL](#_bookmark521), but:

* If no upper layer modules are configured for *transceiver mode indications* using

<User\_TrcvModeIndication>(), no *transceiver mode indication* needs to be configured. [CanIfDispatchUserTrcvModeIndicationUL](#_bookmark521) needs not to be configured.

* If transceivers are not supported ([CanIfTrcvDrvCfg](#_bookmark532) is not configured, [Can-](#_bookmark521) [IfDispatchUserTrcvModeIndicationUL](#_bookmark521) is not configurable.

If no upper layer modules are configured for *state transition notifications* using <User\_TrcvModeIndication>(), no *state transition notification* needs to be con- figured.

**[SWS\_CANIF\_00696]** *[*Configuration of <User\_TrcvModeIndication>(): The name of <User\_TrcvModeIndication>() which will be called by [CanIf](#_bookmark8) shall be configured for [CanIf](#_bookmark8) by parameter [CanIfDispatchUserTrcvModeIndication-](#_bookmark520) [Name](#_bookmark520). This is only necessary if *state transition notifications* are configured via [Can-](#_bookmark521) [IfDispatchUserTrcvModeIndicationUL](#_bookmark521).*]()*

**[SWS\_CANIF\_00697]** *[*Configuration of <User\_TrcvModeIndication>(): If [CanIfDispatchUserTrcvModeIndicationUL](#_bookmark521) is set to CAN\_SM, [CanIfDis-](#_bookmark520) [patchUserTrcvModeIndicationName](#_bookmark520) must be CanSM\_TransceiverMod- eIndication.*]()*

**[SWS\_CANIF\_00698]** *[*Configuration of <User\_TrcvModeIndication>(): If [Can-](#_bookmark521) [IfDispatchUserTrcvModeIndicationUL](#_bookmark521) is set to CDD the name of the API has to be configured via parameter [CanIfDispatchUserTrcvModeIndicationName](#_bookmark520).*] ()*

### Sequence diagrams

The following sequence diagrams show the interactions between [CanIf](#_bookmark8) and [CanDrv](#_bookmark5).

#### Transmit request (single CAN Driver)



«Peripheral»

CanController

CanIf\_Transmit(Std\_ReturnType, PduIdType, const PduInfoType\*)

Can\_Write(Std\_ReturnType, Can\_HwHandleType, const Can\_PduType\*)

alt CAN Controller

Copy L-PDU into CAN Hardware()

[CAN controller hardware object is free]

Copy L-PDU into CAN Hardware()

Can\_Write()

[CAN controller hardware object is busy]

Can\_Write()

Insert L-PDU in transmit buffer()

CanIf\_Transmit()

«module» Can

«module» SchM

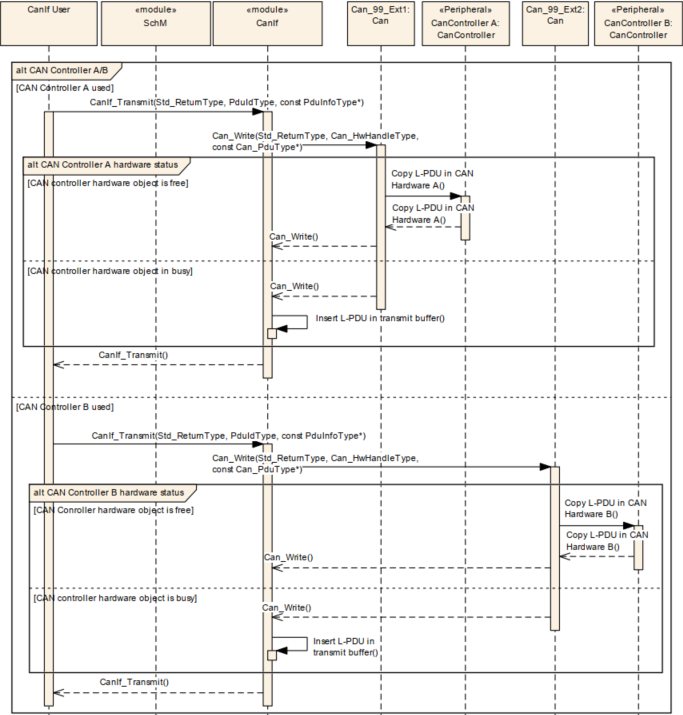
CanIf User

«module» CanIf

**Figure 9.1: Transmission request with a single CAN Driver**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Transmission request** | The upper layer initiates a transmit request via the service  [CanIf\_Transmit()](#_bookmark279). The parameter CanTxPduId identifies the requested [L-SDU](#_bookmark4). The service performs following steps:   * validation of the input parameter * definition of the [CAN Controller](#_bookmark24) to be used   The second parameter \*PduInfoPtr is a pointer on the structure with transmit [L-SDU](#_bookmark4) related data such as SduLength and  \*SduDataPtr. |
| **Start transmission** | [CanIf\_Transmit()](#_bookmark279) requests a transmission and calls the  [CanDrv](#_bookmark5) service Can\_Write() with corresponding processing of  the [HTH](#_bookmark19). |
| **Hardware request** | Can\_Write() writes all [L-PDU](#_bookmark3) data in the [CAN Hardware](#_bookmark25) (if it is  free) and sets the hardware request for transmission. |
| **E\_OK from Can\_Write**  **service** | Can\_Write() returns E\_OK to [CanIf\_Transmit()](#_bookmark279). |
| **CAN\_BUSY from Can\_Write**  **service** | If [CanDrv](#_bookmark5) detects, there are no free hardware objects available, it  returns CAN\_BUSY to [CanIf](#_bookmark8). |
| **Copying into the buffer** | The [L-PDU](#_bookmark3) of the rejected transmit request will be inserted in the  transmit buffer of [CanIf](#_bookmark8) until the next transmit confirmation. |
| **E\_OK from** [**CanIf**](#_bookmark8) | [CanIf\_Transmit()](#_bookmark279) returns E\_OK to the upper layer. |

#### Transmit request (multiple CAN Drivers)

****

**Figure 9.2: Transmission request with multiple CAN Drivers**

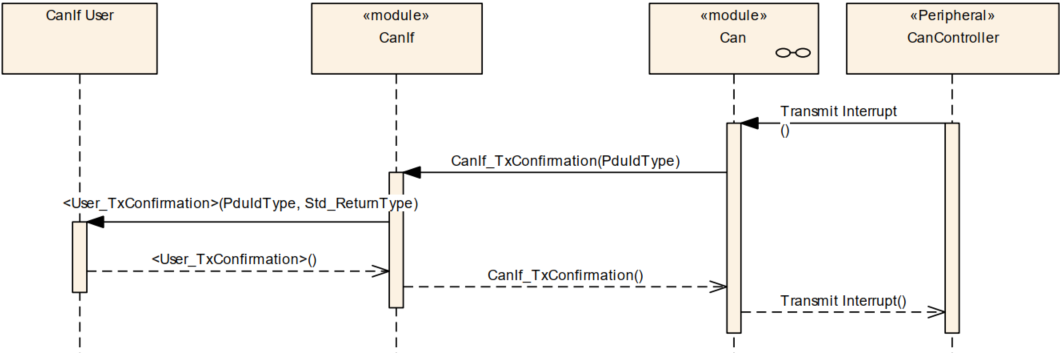
First transmit request:

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Transmission request A** | The upper layer initiates a transmit request via the service  [CanIf\_Transmit()](#_bookmark279). The parameter CanTxPduId identifies the requested [L-SDU](#_bookmark4). The service performs following steps:   * validation of the input parameter * definition of the [CAN Controller](#_bookmark24) to be used (here: Can\_99\_Ext1)   The second parameter \*PduInfoPtr is a pointer on the structure with transmit [L-SDU](#_bookmark4) related data such as SduLength and  \*SduDataPtr. |
| **Start transmission** | [CanIf\_Transmit()](#_bookmark279) requests a transmission and calls the  [CanDrv](#_bookmark5) Can\_99\_Ext1 service Can\_Write\_99\_Ext1() with  corresponding processing of the [HTH](#_bookmark19). |
| **Hardware request** | Can\_Write\_99\_Ext1() writes all [L-PDU](#_bookmark3) data in the [CAN](#_bookmark25)  [Hardware](#_bookmark25) of Controller A (if it is free) and sets the hardware request for transmission. |
| **E\_OK from Can\_Write**  **service** | Can\_Write\_99\_Ext1() returns E\_OK to [CanIf\_Transmit()](#_bookmark279). |
| **CAN\_BUSY from Can\_Write**  **service** | If [CanDrv](#_bookmark5) Can\_99\_Ext1 detects, there are no free hardware  objects available, it returns CAN\_BUSY to [CanIf](#_bookmark8). |
| **Copying into the buffer** | The [L-PDU](#_bookmark3) of the rejected transmit request will be inserted in the  transmit buffers of [CanIf](#_bookmark8) until the next transmit confirmation. |
| **E\_OK from** [**CanIf**](#_bookmark8) | [CanIf\_Transmit()](#_bookmark279) returns E\_OK to the upper layer. |

Second transmit request:

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Transmission request B** | The upper layer initiates a transmit request via the service  [CanIf\_Transmit()](#_bookmark279). The parameter CanTxPduId identifies the requested [L-SDU](#_bookmark4). The service performs following steps:   * validation of the input parameter * definition of the [CAN Controller](#_bookmark24) to be used (here: Can\_99\_Ext2)   The second parameter \*PduInfoPtr is a pointer on the structure with transmit [L-SDU](#_bookmark4) related data such as SduLength and  \*SduDataPtr. |
| **Start transmission** | [CanIf\_Transmit()](#_bookmark279) starts a transmission and calls the [CanDrv](#_bookmark5)  Can\_99\_Ext2 service Can\_Write\_99\_Ext2() with  corresponding processing of the [HTH](#_bookmark19). |
| **Hardware request** | Can\_Write\_99\_Ext2() writes all [L-PDU](#_bookmark3) data in the [CAN](#_bookmark25)  [Hardware](#_bookmark25) of Controller B (if it is free) and sets the hardware request for transmission. |
| **E\_OK from Can\_Write**  **service** | Can\_Write\_99\_Ext2() returns E\_OK to [CanIf\_Transmit()](#_bookmark279). |
| **CAN\_BUSY from Can\_Write**  **service** | If [CanDrv](#_bookmark5) Can\_99\_Ext2 detects, there are no free hardware  objects available, it returns CAN\_BUSY to [CanIf](#_bookmark8). |
| **Copying into the buffer** | The [L-PDU](#_bookmark3) of the rejected transmit request will be inserted in the  transmit buffers of [CanIf](#_bookmark8) until the next transmit confirmation. |
| **E\_OK from** [**CanIf**](#_bookmark8) | [CanIf\_Transmit()](#_bookmark279) returns E\_OK to the upper layer. |

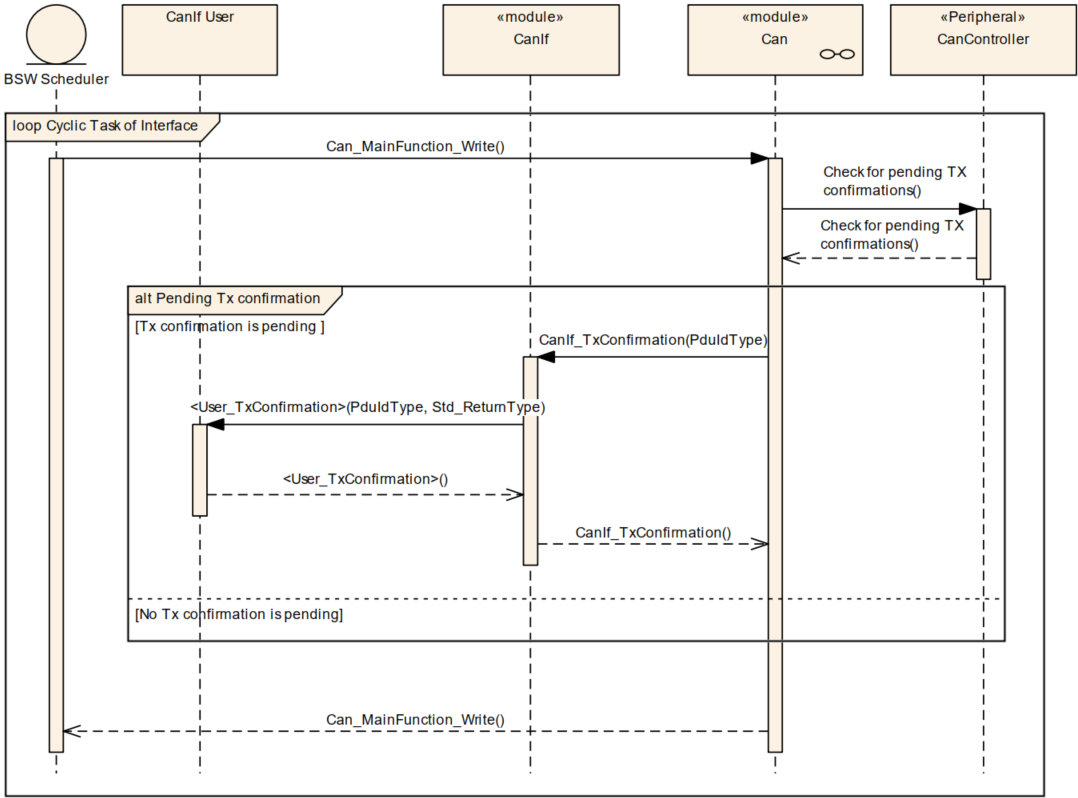
#### Transmit confirmation (interrupt mode)

****

**Figure 9.3: Transmit confirmation interrupt driven**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Transmit interrupt** | The acknowledged CAN frame signals a successful transmission to  the receiving [CAN Controller](#_bookmark24) and triggers the transmit interrupt. |
| **Confirmation to** [**CanIf**](#_bookmark8) | [CanDrv](#_bookmark5) calls the service [CanIf\_TxConfirmation()](#_bookmark377). The  parameter CanTxPduId specifies the [L-PDU](#_bookmark3) previously sent by  Can\_Write().  [CanDrv](#_bookmark5) must store the all in [HTHs](#_bookmark19) pending [L-PDU](#_bookmark3) Ids in an array organized per [HTH](#_bookmark19) to avoid new search of the [L-PDU](#_bookmark3) ID for call of  [CanIf\_TxConfirmation()](#_bookmark377). |
| **Confirmation to upper layer** | Calling of the corresponding upper layer confirmation service  <User\_TxConfirmation>(id, E\_OK). It signals a successful  [L-SDU](#_bookmark4) transmission to the upper layer. |

#### Transmit confirmation (polling mode)

****

**Figure 9.4: Transmit confirmation polling driven**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Cyclic Task** [**CanDrv**](#_bookmark5) | The service Can\_MainFunction\_Write() is called by the BSW  Scheduler. |
| **Check for pending transmit**  **confirmations** | Can\_MainFunction\_Write() checks the underlying [CAN](#_bookmark24)  [Controller](#_bookmark24)(s) about pending transmit confirmations of previously succeeded transmit events. |
| **Transmit Confirmation** | The acknowledged CAN frame signals a successful transmission  to the sending [CAN Controller](#_bookmark24). |
| **Confirmation to** [**CanIf**](#_bookmark8) | [CanDrv](#_bookmark5) calls the service [CanIf\_TxConfirmation()](#_bookmark377). The  parameter CanTxPduId specifies the [L-PDU](#_bookmark3) previously sent by  Can\_Write().  [CanDrv](#_bookmark5) must store the all in [HTHs](#_bookmark19) pending [L-PDU](#_bookmark3) Ids in an array organized per [HTH](#_bookmark19) to avoid new search of the [L-PDU](#_bookmark3) ID for call of  [CanIf\_TxConfirmation()](#_bookmark377). |
| **Confirmation to upper layer** | Calling of the corresponding upper layer confirmation service  <User\_TxConfirmation>(id, E\_OK). It signals a successful  [L-SDU](#_bookmark4) transmission to the upper layer. |

#### Transmit confirmation (with buffering)



«module»

SchM

«module»

CanIf

«module»

Can

«Peripheral»

CanController

Transmit Confirmation Interrupt()

CanIf\_TxConfirmation(PduIdType)

check transmit buffers for other pending L-PDU()

alt Transmit Buffering

[Buffer is filled]

Can\_Write(Std\_ReturnType, Can\_HwHandleType, const Can\_PduType\*)

Write L-PDU into CAN Hardware()

Write L-PDU into CAN Hardware()

Can\_Write()

Remove L-PDU successfully requested for transmission from transmit buffer()

[Buffer is empty]

<User\_TxConfirmation>(PduIdType, Std\_ReturnType)

<User\_TxConfirmation>()

CanIf\_TxConfirmation()

Transmit Confirmation Interrupt()

CanIf User

**Figure 9.5: Transmit confirmation with buffering**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Transmit interrupt** | Acknowledged CAN frame signals successful transmission to  receiving [CAN Controller](#_bookmark24) and triggers transmit interrupt. |
| **Confirmation to** [**CanIf**](#_bookmark8) | [CanDrv](#_bookmark5) calls service [CanIf\_TxConfirmation()](#_bookmark377). Parameter  CanTxPduId specifies the [L-PDU](#_bookmark3) previously transmitted by Can\_Write(). [CanDrv](#_bookmark5) must store the all in [HTHs](#_bookmark19) pending [L-PDU](#_bookmark3) Ids in an array organized per [HTH](#_bookmark19) to avoid new search of the  [L-PDU](#_bookmark3) ID for call of [CanIf\_TxConfirmation()](#_bookmark377). |
| **Check of transmit buffers** | The transmit buffers of [CanIf](#_bookmark8) checked, whether a pending [L-PDU](#_bookmark3)  is stored or not. |
| **Transmit request passed to**  [**CanDrv**](#_bookmark5) | In case of pending [L-PDUs](#_bookmark3) in the transmit buffers the highest  priority order the latest [L-PDU](#_bookmark3) is requested for transmission by  Can\_Write(). It signals a successful [L-PDU](#_bookmark3) transmission to the  upper layer. Thus Can\_Write() can be called re-entrant. |
| **Remove transmitted** [**L-PDU**](#_bookmark3)  **from transmit buffers** | The [L-PDU](#_bookmark3) pending for transmission is removed from the  transmission buffers by [CanIf](#_bookmark8). |
| **Confirmation to the upper**  **layer** | Calling of the corresponding upper layer confirmation service  <User\_TxConfirmation>(id, E\_OK). It signals a successful  [L-SDU](#_bookmark4) transmission to the upper layer. |

#### Trigger Transmit Request



CanIf\_Transmit(Std\_ReturnType, PduIdType, const PduInfoType\*)

Can\_Write(Std\_ReturnType,

Can\_HwHandleType, const Can\_PduType\*)

alt Controller HW Status

[Controller HW object free]

CanIf\_TriggerTransmit(Std\_ReturnType, PduIdType, PduInfoType\*)

<CanIfUser>\_CanIfTriggerTransmit(Std\_ReturnT e, PduIdType, PduInfoType\*)

Copy L-Pdu into CAN hardware

Copy L-PDU into CAN hardware

CanIf\_TriggerTransmit()

Can\_Write()

Can\_Write()

CanIf\_Transmit()

[Controller HW object busy]

yp

«module» Can

«module» CanIf

CanIfUser

«peripheral» CanController

**Figure 9.6: Trigger Transmit Request**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Transmission request** | The upper layer initiates a transmit request via the service  [CanIf\_Transmit()](#_bookmark279). The parameter CanTxPduId identifies the requested [L-SDU](#_bookmark4). The service performs following steps:   * validation of the input parameter * definition of the [CAN Controller](#_bookmark24) to be used   The second parameter \*PduInfoPtr is a pointer to the structure with the size (SduLength) of the [L-SDU](#_bookmark4) to be transmitted. The actual SDU data has not been passed by the upper layer. Hence, the pointer \*SduDataPtr points to NULL. |
| **Start transmission** | [CanIf\_Transmit()](#_bookmark279) requests a transmission and calls the  [CanDrv](#_bookmark5) service Can\_Write() with corresponding processing of  the [HTH](#_bookmark19). |
| **Trigger transmission** | If the CAN hardware is free Can\_Write() requests the SDU data  from [CanIf](#_bookmark8) by its service [CanIf\_TriggerTransmit()](#_bookmark374) passing the [L-SDUs](#_bookmark4) corresponding ID and a pointer to the CAN hardware’s buffer. [CanIf](#_bookmark8) forwards the trigger transmit request to the corresponding upper layer (CanIfUser). [CanIf](#_bookmark8) passes the buffer pointer received by [CanDrv](#_bookmark5). The CanIfUser finally copies the SDU data to the buffer provided by [CanIf](#_bookmark8) (the CAN hardware buffer) and returns status and number of bytes effectively written. |
| **E\_OK from Can\_Write()**  **service** | Can\_Write() returns E\_OK to [CanIf\_Transmit()](#_bookmark279). |
| **CAN\_BUSY from**  **Can\_Write() service** | If [CanDrv](#_bookmark5) detects, there are no free hardware objects available, it  returns CAN\_BUSY to [CanIf](#_bookmark8). |
| **Queuing of transmission**  **request** | The [Transmit Request](#_bookmark33) for the [L-PDU](#_bookmark3), which has been rejected  by [CanDrv](#_bookmark5), is queued by [CanIf](#_bookmark8) until the next transmit confirmation. |
| **E\_OK from** [**CanIf**](#_bookmark8) | [CanIf\_Transmit()](#_bookmark279) returns E\_OK to the upper layer. |

#### Receive indication (interrupt mode)



CanIf User

«module»

CanIf

«module»

Can

«Peripheral»

CanController

Receive Interrupt()

Invalidation of hardware object() Invalidation of hardware object()

alt Temporary buffer usage

[Temp. buffer used = Data normalization necessary]

Copy received L-PDU into temporary buffer()

Copy received L-PDU into temporary buffer()

[Temp. buffer not used = Data normalization not necessary]

CanIf\_RxIndication(const Can\_HwType\*, const PduInfoType\*)

Software filtering (optional) and L-PDU assignment

[CAN L-PDU ID was found]: Data Length Check (optional)

<User\_RxIndication>(PduIdType, const PduInfoType\*)

alt Temporary buffer usage

[Temp. buffer used = Data normalization necessary]

Copy Data()

Copy Data()

[Temp. buffer not used = Data normalization not necessary]

Copy Data()

Copy Data()

<User\_RxIndication>()

CanIf\_RxIndication()

Validation of hardware object()

Validation of hardware object() Receive

Interrupt()

**Figure 9.7: Receive indication interrupt driven**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Receive Interrupt** | The [CAN Controller](#_bookmark24) indicates a successful reception and  triggers a receive interrupt. |
| **Invalidation of CAN**  **hardware object, provide CPU access to CAN mailbox** | The CPU ([CanDrv](#_bookmark5)) get exclusive access rights to the CAN mailbox  or at least to the corresponding hardware object, where new data were received. |

|  |  |
| --- | --- |
| **Buffering, normalizing** | The [L-PDU](#_bookmark3) is normalized and is buffered in the temporary buffer  located in [CanDrv](#_bookmark5). Each [CanDrv](#_bookmark5) owns such a temporary buffer for every [Physical Channel](#_bookmark32) only if normalizing of the data is necessary. |
| **Indication to** [**CanIf**](#_bookmark8) | The reception is indicated to [CanIf](#_bookmark8) by calling of  [CanIf\_RxIndication()](#_bookmark383). The [HRH](#_bookmark18) specifies the CAN RAM [Hardware Object](#_bookmark29) and the corresponding [CAN Controller](#_bookmark24), which contains the received [L-PDU](#_bookmark3). The temporary buffer is  referenced to [CanIf](#_bookmark8) by PduInfoPtr->SduDataPtr. |
| **Software Filtering** | The Software Filtering checks, whether the received [L-PDU](#_bookmark3) will be  processed on a local ECU. If not, the received [L-PDU](#_bookmark3) is not indicated to upper layers. Further processing is suppressed. |
| **Data Length Check** | If the [L-PDU](#_bookmark3) is found, the Data Length of the received [L-PDU](#_bookmark3) is  compared with the expected, statically configured one for the  received [L-PDU](#_bookmark3). |
| **Receive Indication to the**  **upper layer** | The corresponding receive indication service of the upper layer is  called. This signals a successful reception to the target upper layer. The parameter RxPduId specifies the [L-SDU](#_bookmark4), the second parameter is the reference on the temporary buffer within the  [L-SDU](#_bookmark4).  During is execution of this service the CAN hardware buffers must  be unlocked for CPU access/locked for [CAN Controller](#_bookmark24) access. |
| **Validation of CAN hardware**  **object, allow access of** [**CAN**](#_bookmark24)[**Controller**](#_bookmark24) **to CAN mailbox** | The [CAN Controller](#_bookmark24) 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. |

#### Receive indication (polling mode)



«Peripheral»

CanController

BSW Scheduler

loop Cyclic Task of Interface

Can\_MainFunction\_Read()

Check for pending Rx indication()

alt Pending Rx indication

[Rx indication pending]

Invalidation of hardware object()

alt Temporary buffer usage

[Temp. buffer used = Data normalization necessary]

Copy received L-PDU into temporary buffer()

[Temp. buffer not used = Data normalization not necessary]

CanIf\_RxIndication(const Can\_HwType\*, const PduInfoType\*)

Software filtering (optional) and L-PDU assignment

«module»

CanIf

<User\_RxIndication>(PduIdType, const PduInfoType\*)

alt Temporary buffer usage

[Temp. buffer used = Data normalization necessary]

Copy data()

[Temp. buffer not used = Data normalization not necessary]

Copy data()

<User\_RxIndication>()

CanIf\_RxIndication()

Validation of hardware object()

[No Rx indication pending]

Can\_MainFunction\_Read()

CanIf User

«module»

Can

[CAN L-PDU ID was found]: Data Length Check (optional)

**Figure 9.8: Receive indication polling driven**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Cyclic Task** [**CanDrv**](#_bookmark5) | The service Can\_MainFunction\_Read() is called by the BSW  Scheduler. |
| **Check for new received**  [**L-PDU**](#_bookmark3) | Can\_MainFunction\_Read() checks the underlying [CAN](#_bookmark24)  [Controller](#_bookmark24)(s) about new received [L-PDUs](#_bookmark3). |
| **Invalidation of CAN**  **hardware object, provide CPU access to CAN mailbox** | In case of a new receive event the CPU ([CanDrv](#_bookmark5)) get exclusive  access rights to the CAN mailbox or at least to the corresponding hardware object, where new data were received. |
| **Buffering, normalizing** | In case of a new receive event the [L-PDU](#_bookmark3) is normalized and is  buffered in the temporary buffer located in [CanDrv](#_bookmark5). Each [CanDrv](#_bookmark5) owns such a temporary buffer for every [Physical Channel](#_bookmark32) only if normalizing of the data is necessary. |
| **Indication to** [**CanIf**](#_bookmark8) | The reception is indicated to [CanIf](#_bookmark8) by calling of  [CanIf\_RxIndication()](#_bookmark383). The [HRH](#_bookmark18) specifies the CAN RAM [Hardware Object](#_bookmark29) and the corresponding [CAN Controller](#_bookmark24), which contains the received [L-PDU](#_bookmark3). The temporary buffer is  referenced to [CanIf](#_bookmark8) by PduInfoPtr->SduDataPtr. |
| **Software Filtering** | The Software Filtering checks, whether the received [L-PDU](#_bookmark3) will be  processed on a local ECU. If not, the received [L-PDU](#_bookmark3) is not indicated to upper layers. Further processing is suppressed. |
| **Data Length Check** | If the [L-PDU](#_bookmark3) is found, the Data Length of the received [L-PDU](#_bookmark3) is  compared with the expected, statically configured one for the  received [L-PDU](#_bookmark3). |
| **Receive Indication to the**  **upper layer** | If configured, the corresponding receive indication service of the  upper layer is called. This signals a successful reception to the target upper layer. The parameter RxPduId specifies the [L-SDU](#_bookmark4), the second parameter is the reference on the temporary buffer within the [L-SDU](#_bookmark4).  During is execution of this service the CAN hardware buffers must  be unlocked for CPU access/locked for [CAN Controller](#_bookmark24) access. |
| **Validation of CAN hardware**  **object, allow access of** [**CAN**](#_bookmark24)[**Controller**](#_bookmark24) **to CAN mailbox** | The [CAN Controller](#_bookmark24) get back exclusive access rights to the  CAN mailbox or at least to the corresponding hardware object, where new data were already being copied into the upper layer  buffer. |

#### Read received data

CanIf User



«module»

Can

«Peripheral»

CanController

Receive Interrupt()

Invalidation of hardware object()

Invalidation of hardware object()

CanIf\_RxIndication(const Can\_HwType\*, const PduInfoType\*)

[L-PDU reception in BasicCAN]: Software filtering and L-PDU assignment()

[CAN L-PDU ID was found]:

Data Length Check()

Copy data to CANIF receive L-PDU buffer()

Copy data to CANIF receive L-PDU buffer()

Set Indication Flag()

<User\_RxIndication>(PduIdType, const PduInfoType\*)

<User\_RxIndication>()

CanIf\_RxIndication()

Validation of hardware object()

Validation of hardware object()

Receive Interrupt()

CanIf\_ReadRxNotifStatus(CanIf\_NotifStatusType, PduIdType)

Read Indication flag()

Reset Indication flag()

CanIf\_ReadRxNotifStatus()

CanIf\_ReadRxPduData(Std\_ReturnType, PduIdType, PduInfoType\*\*)

Read data from CANIF Rx buffer()

CanIf\_ReadRxPduData()

«module» CanIf

«module» SchM

**Figure 9.9: Read received data**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **Receive Interrupt** | The [CAN Controller](#_bookmark24) indicates a successful reception and  triggers a receive interrupt. |
| **Invalidation of CAN**  **hardware object, provide CPU access to CAN mailbox** | The CPU ([CanDrv](#_bookmark5)) get exclusive access rights to the CAN mailbox  or at least to the corresponding hardware object, where new data were received. |

|  |  |
| --- | --- |
| **Buffering, normalizing** | The [L-PDU](#_bookmark3) is normalized and is buffered in the temporary buffer  located in [CanDrv](#_bookmark5). Each [CanDrv](#_bookmark5) owns such a temporary buffer for every [Physical Channel](#_bookmark32) only if normalizing of the data is necessary. |
| **Indication to** [**CanIf**](#_bookmark8) | The reception is indicated to [CanIf](#_bookmark8) by calling of  [CanIf\_RxIndication()](#_bookmark383). The [HRH](#_bookmark18) specifies the CAN RAM [Hardware Object](#_bookmark29) and the corresponding [CAN Controller](#_bookmark24), which contains the received [L-PDU](#_bookmark3). The temporary buffer is  referenced to [CanIf](#_bookmark8) by PduInfoPtr->SduDataPtr. |
| **Software Filtering** | The Software Filtering checks, whether the received [L-PDU](#_bookmark3) will be  processed on a local ECU. If not, the received [L-PDU](#_bookmark3) is not indicated to upper layers. Further processing is suppressed. |
| **Data Length Check** | If the [L-PDU](#_bookmark3) is found, the Data Length of the received [L-PDU](#_bookmark3) is  compared with the expected, statically configured one for the  received [L-PDU](#_bookmark3). |
| **Copy data** | The data is copied out of the CAN hardware into the receive [CAN](#_bookmark3)  [L-PDU](#_bookmark3) buffers in [CanIf](#_bookmark8). During access the CAN hardware buffers must be unlocked for CPU access/locked for [CAN Controller](#_bookmark24) access. |
| **Indication Flag** | Set indication status flag for the received [L-PDU](#_bookmark3) in [CanIf](#_bookmark8). |
| **Receive Indication to the**  **upper layer** | The corresponding receive indication service of the upper layer is  called. This signals a successful reception to the target upper layer. The parameter RxPduId specifies the [L-SDU](#_bookmark4), the second parameter is the reference on the temporary buffer within the  [L-SDU](#_bookmark4). |
| **Validation of CAN hardware**  **object, allow access of** [**CAN**](#_bookmark24)[**Controller**](#_bookmark24) **to CAN mailbox** | The [CAN Controller](#_bookmark24) get back exclusive access rights to the  CAN mailbox or at least to the corresponding hardware object, where new data were already being copied into the upper layer  buffer. |
| **Read indication status** | Times later the upper layer can read the indication status by call of  [CanIf\_ReadRxNotifStatus()](#_bookmark299). This service can also be used for transmit [L-PDUs](#_bookmark3). Then it return the confirmation status. |
| **Reset indication status** | Before [CanIf\_ReadRxNotifStatus()](#_bookmark299) returns, the indication  status is reset. |
| **Read received data** | Times later the upper layer can read the received data by call of  [CanIf\_ReadRxPduData()](#_bookmark289). |
| **Read CanIf Rx buffer** | [CanIf\_ReadRxPduData()](#_bookmark289) reads the data from [CanIf](#_bookmark8) Rx buffer. |
| **E\_OK from** [**CanIf**](#_bookmark8) | If [CanIf\_ReadRxPduData()](#_bookmark289) was successful, the request returns  E\_OK with valid PduInfoPtr. |

#### Start CAN network



CanIf User

«module»

CanIf

«module»

Can

«Peripheral»

CanController

loop Requesting CAN controller mode consecutively. If mode changed -> CanIf\_ControllerModeIndication()

Can\_MainFunction\_Mode()

CanIf\_SetControllerMode(Std\_ReturnType, uint8, Can\_ControllerStateType)

Can\_SetControllerMode(Std\_ReturnType, uint8, Can\_ControllerStateType)

Disable Wakeup

interrupt, if supported()

request CAN controller mode transition to START()

alt CAN Controller Mode

[STOPPED] CanIf\_SetControllerMode returns with E\_OK()

Can\_SetControllerMode returns with E\_OK()

CAN controller mode changes to START

CanIf\_ControllerModeIndication(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>()

CanIf\_ControllerModeIndication()

[STOPPED with direct indication]

CanIf\_ControllerModeIndication(Controller, ControllerMode)

<User\_ControllerModeIndication>(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>()

CanIf\_ControllerModeIndication()

Can\_SetControllerMode returns with E\_OK()

CanIf\_ControllerMode returns with E\_OK()

[STARTED]

CanIf\_ControllerModeIndication(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>()

CanIf\_ControllerModeIndication()

Can\_SetControllerMode returns with E\_OK()

CanIf\_SetControllerMode returns with E\_OK()

[SLEEP]

Can\_SetControllerMode returns with E\_NOT\_OK()

CanIf\_SetControllerMode returns with E\_NOT\_OK()

**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](#_bookmark402) [(Controller, ControllerMode)](#_bookmark402). |
| **The upper layer requests "**  **STARTED" mode of the desired CAN controller** | The upper layer calls [CanIf\_SetControllerMode](#_bookmark264)  [(ControllerId, CAN\_CS\_STARTED)](#_bookmark264) to request STARTED  mode for the requested CAN controller. |
| **CanDrv disables wake up**  **interrupts, if supported** | This is only done in case of requesting "STARTED" mode. If "  SLEEP" mode of CAN controller is requested, here the wake up  interrupts are enabled. In case of "STOPPED", nothing happens. |
| **CanDrv requests the CAN**  **controller to transition into the requested mode ( CAN\_CS\_STARTED).** | During function call Can\_SetControllerMode(Controller,  Can\_ControllerStateType), the CanDrv enters the request into the hardware of the CAN controller. This may mean that the controller mode transitions directly, but it could mean that it takes a few milliseconds until the controller changes its state. It depends on the controllers. |
| The following reaction depends on the controller and its current operation mode | |
| **CAN controller was in**  **STOPPED mode** | The former request Can\_SetControllerMode() returns and  informs CanIf about a successful request which in turn returns the upper layer request [CanIf\_SetControllerMode()](#_bookmark264). The Can\_MainFunction\_Mode() detects the successful mode transition of the CAN controller and inform the CanIf  asynchronously via [CanIf\_ControllerModeIndication](#_bookmark402) [(Controller, CAN\_CS\_STARTED)](#_bookmark402). |
| **CAN controller was in**  **STOPPED mode and the CAN controller transitions very fast so that mode indication is called during transition request** | During the former request Can\_SetControllerMode() the  function [CanIf\_ControllerModeIndication(Controller,](#_bookmark402) [CAN\_CS\_STARTED)](#_bookmark402) is called to inform the CanIf directly about the successful mode transition. When  [CanIf\_ControllerModeIndication(Controller,](#_bookmark402) [CAN\_CS\_STARTED)](#_bookmark402) returned, the request Can\_SetControllerMode() returns and informs CanIf about a successful request which in turn returns the upper layer request [CanIf\_SetControllerMode()](#_bookmark264). |
| **CAN controller was in**  **STARTED mode** | During the former request Can\_SetControllerMode() the  function [CanIf\_ControllerModeIndication(Controller,](#_bookmark402) [CAN\_CS\_STARTED)](#_bookmark402) is called to inform the CanIf directly about the successful mode transition (because the mode was already started). When [CanIf\_ControllerModeIndication](#_bookmark402) [(Controller, CAN\_CS\_STARTED)](#_bookmark402) returned, the request Can\_SetControllerMode() returns and informs CanIf about a successful request which in turn returns the upper layer request  [CanIf\_SetControllerMode()](#_bookmark264). |
| **CAN controller was in**  **SLEEP mode** | This transition is not allowed -> E\_NOT\_OK. |

#### BusOff notification



«module»

Can

BusOff Detection()

Set CAN Controller to STOPPED mode, if necessary()

Set CAN Controller to STOPPED mode, if necessary()

CanIf\_ControllerBusOff(uint8)

<User\_ControllerBusOff>(uint8)

Reset transmit queue()

<User\_ControllerBusOff>()

CanIf\_ControllerBusOff()

BusOff Detection()

«Peripheral» CanController

«module» CanIf

<User\_ControllerBusOff>

**Figure 9.11: BusOff notification**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **BusOff detection interrupt** | The CAN controller signals a BusOff event. |
| **Stop CAN controller** | CAN controller is set to STOPPED mode by the CAN Driver, if  necessary. |
| **BusOff indication to CAN**  **Interface** | BusOff is notified to the CanIf by calling of  [CanIf\_ControllerBusOff()](#_bookmark389) |
| **BusOff indication to upper**  **layer (CanSM)** | BusOff is notified to the upper layer by calling of  <User\_ControllerBusOff>() |

#### BusOff recovery



<User\_ControllerBusOff>

«module»

CanIf

«module»

Can

«Peripheral»

CanController

loop Requesting CAN controller mode consecutively. If mode changed -> CanIf\_ControllerModeIndication().

Can\_MainFunction\_Mode()

opt CAN controller

[BUSSOFF]

BusOff Detection()

Set CAN controller to STOPPED mode, if necessary() Set CAN controller to STOPPED mode, if necessary()

[STOPPED]

CanIf\_ControllerBusOff(uint8)

Reset transmit queue()

<User\_ControllerBusOff>(uint8)

<User\_ControllerBusOff>()

CanIf\_ControllerBusOff()

BusOff Detection()

CanIf\_SetControllerMode(Std\_ReturnType, uint8, Can\_ControllerStateType)

Can\_SetControllerMode(Std\_ReturnType, uint8, Can\_ControllerStateType)

Reset CAN controller, if necessary()

request CAN controller mode transition to START()

Can\_SetContollerMode()

CanIf\_SetControllerMode()

CanIf\_ControllerModeIndication(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>(uint8, Can\_ControllerStateType)

<User\_ControllerModeIndication>()

CanIf\_ControllerModeIndication()

**Figure 9.12: BusOff recovery**

|  |  |
| --- | --- |
| **Activity** | **Description** |
| **BusOff detection interrupt** | The CAN controller signals a BusOff event. |
| **Stop CAN controller** | CAN controller is set to STOPPED mode by the [CanDrv](#_bookmark5), if necessary |
| **BusOff indication to** [**CanIf**](#_bookmark8) | BusOff is notified to the [CanIf](#_bookmark8) by calling of [CanIf\_ControllerBusOff()](#_bookmark389). The transmit buffers inside [CanIf](#_bookmark8) will be reset. |
| **BusOff indication to upper layer** | BusOff is notified to the upper layer by calling of <User\_ControllerBusOff>() |
| **Upper Layer (CanSM) initiates BusOff Recovery** | After a time specified by the BusOff Recovery algorithm the  Recovery process itself in initiated by [CanIf\_SetControllerMode(ControllerId,](#_bookmark264) [CAN\_CS\_STARTED)](#_bookmark264). |
| **Restart of CAN controller** | The driver restarts the CAN controller by call of Can\_SetControllerMode(Controller, CAN\_CS\_STARTED). |
| **CAN controller started** | [CanDrv](#_bookmark5) informs [CanIf](#_bookmark8) about the successful start by calling [CanIf\_ControllerModeIndication()](#_bookmark402). [CanIf](#_bookmark8) informs in turn upper layers about the mode change. |

### Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. For general information about the definition of containers and param- eters, refer to the [[9](#_bookmark44), chapter 10.1 "Introduction to configuration specification" in SWS\_BSWGeneral].

[section](#_bookmark452) [10.1](#_bookmark452) specifies the structure (containers) and the parameters of the CanIf.

#### Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe [chapter](#_bookmark140) [7](#_bookmark140) “[Functional specification](#_bookmark140)” and [chapter](#_bookmark246) [8](#_bookmark246) “[API](#_bookmark246) [specification](#_bookmark246)”.

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](#_bookmark8).

The CanIf has access to the [CanDrv](#_bookmark5) configuration data. All public CanDrv configura- tion data are described in [[1](#_bookmark36), Specification of CAN Driver].



+container

+container

+subContainer

+subContainer

+subContainer

+container

+subContainer

+container

+container

+subContainer

lowerMultiplicity = 1 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfTrcvCfg: EcucParamConfContainerDef

CanIfTrcvDrvCfg: EcucParamConfContainerDef

lowerMultiplicity = 1 upperMultiplicity = \*

upperMultiplicity = \* lowerMultiplicity = 1

CanIfCtrlDrvCfg: EcucParamConfContainerDef

CanIfCtrlCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfDispatchCfg: EcucParamConfContainerDef

CanIfHrhRangeCfg: EcucParamConfContainerDef

CanIfTxPduCfg: EcucParamConfContainerDef

+subContainer

+subContainer

lowerMultiplicity = 0 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfRxPduCfg: EcucParamConfContainerDef

lowerMultiplicity = 1 upperMultiplicity = 1

CanIfHrhCfg: EcucParamConfContainerDef

+subContainer

CanIfInitCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

upperMultiplicity = 1 lowerMultiplicity = 1

+container

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfInitHohCfg: EcucParamConfContainerDef

CanIfPublicCfg: EcucParamConfContainerDef

CanIfHthCfg: EcucParamConfContainerDef

upperMultiplicity = 1 lowerMultiplicity = 0

CanIfPrivateCfg: EcucParamConfContainerDef

CanIf: EcucModuleDef

**Figure 10.1:** **Overview about CAN Interface configuration containers**

##### CanIf

|  |  |  |
| --- | --- | --- |
| **Module SWS Item** | ECUC\_CanIf\_00244 | |
| **Module Name** | CanIf | |
| **Module Description** | This container includes all necessary configuration sub-containers  according the CAN Interface configuration structure. | |
| **Post-Build Variant**  **Support** | true | |
| **Supported Config**  **Variants** | VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-  COMPILE | |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfCtrlDrvCfg](#_bookmark530) | 1..\* | Configuration parameters for all the underlying CAN  Driver modules are aggregated under this container. For each CAN Driver module a seperate instance of this container has to be provided. |
| [CanIfDispatchCfg](#_bookmark509) | 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. |

|  |  |  |
| --- | --- | --- |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfInitCfg](#_bookmark479) | 1 | This container contains the init parameters of the CAN  Interface. |
| [CanIfPrivateCfg](#_bookmark457) | 1 | This container contains the private configuration  (parameters) of the CAN Interface. |
| [CanIfPublicCfg](#_bookmark461) | 1 | This container contains the public configuration  (parameters) of the CAN Interface. |
| [CanIfTrcvDrvCfg](#_bookmark532) | 0..\* | This container contains the configuration (parameters)  of all addressed CAN transceivers by each underlying CAN Transceiver Driver module. For each CAN transceiver Driver a seperate instance of this container shall be provided. |



+container

+container

+container

+subContaine

+subContainer

+container

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfBufferCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+subContainer

CanIfRxPduCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+subContainer

CanIfTxPduCfg: EcucParamConfContainerDef

+subContainer

lowerMultiplicity = 0 upperMultiplicity = \*

lowerMultiplicity = 1 upperMultiplicity = 1

CanIfInitHohCfg: EcucParamConfContainerDef

CanIfInitCfg: EcucParamConfContainerDef

upperMultiplicity = \* lowerMultiplicity = 1

lowerMultiplicity = 1 upperMultiplicity = \*

CanIfCtrlCfg: EcucParamConfContainerDef

r

CanIfCtrlDrvCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+container

CanIfTrcvDrvCfg: EcucParamConfContainerDef

CanIfDispatchCfg: EcucParamConfContainerDef

+container

CanIfPrivateCfg: EcucParamConfContainerDef

upperMultiplicity = 1 lowerMultiplicity = 1

upperMultiplicity = 1 lowerMultiplicity = 0

CanIfPublicCfg: EcucParamConfContainerDef

CanIf: EcucModuleDef

**Figure 10.2: AR\_EcucDef\_CanIf**

##### CanIfPrivateCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00245] |
| **Container Name** | CanIfPrivateCfg |
| **Parent Container** | [CanIf](#_bookmark455) |
| **Description** | This container contains the private configuration (parameters) of the CAN Interface. |
| **Configuration Parameters** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfFixedBuffer [ECUC\_CanIf\_00827] | | |
| **Parent Container** | [CanIfPrivateCfg](#_bookmark457) | | |
| **Description** | This parameter defines if the buffer element length shall be fixed to 8 Bytes for buffers to which only PDUs < 8 Bytes are assigned.  TRUE: Minimum buffer element length is fixed to 8 Bytes. FALSE: Buffer element length depends on the configured length of the referenced global PDUs (see ECUC\_EcuC\_00078). | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPrivateDataLengthCheck [ECUC\_CanIf\_00617] | | |
| **Parent Container** | [CanIfPrivateCfg](#_bookmark457) | | |
| **Description** | Selects whether Data Length Check is supported.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| true | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPrivateSoftwareFilterType [ECUC\_CanIf\_00619] | | |
| **Parent Container** | [CanIfPrivateCfg](#_bookmark457) | | |
| **Description** | Selects the desired software filter mechanism for reception only. Each implemented software filtering method is identified by this enumeration number.  Range: Types implemented software filtering methods | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | BINARY | Selects Binary Filter method. | |
| INDEX | Selects Index Filter method. | |
| LINEAR | Selects Linear Filter method. | |
| TABLE | Selects Table Filter method. | |
| false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local  dependency: BasicCAN reception must be enabled by referenced parameter CanHandleType of the CAN Driver module via CanIfHrhIdSymRef for at least one HRH. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfSupportTTCAN [ECUC\_CanIf\_00675] | | |
| **Parent Container** | [CanIfPrivateCfg](#_bookmark457) | | |
| **Description** | Defines whether TTCAN is supported.  TRUE: TTCAN is supported. FALSE: TTCAN is not supported, only normal CAN communication is possible. | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| 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. |



+container

upperMultiplicity = 1 lowerMultiplicity = 0

CanIf: EcucModuleDef



+parameter

defaultValue = True

CanIfPrivateDataLengthCheck: EcucBooleanParamDef

+parameter



CanIfPrivateSoftwareFilterType: EcucEnumerationParamDef

CanIfPrivateCfg: EcucParamConfContainerDef

INDEX:

EcucEnumerationLiteralDef



TABLE:

EcucEnumerationLiteralDef

+literal

BINARY:

EcucEnumerationLiteralDef

LINEAR:

EcucEnumerationLiteralDef

+literal

+literal

+literal



+parameter

defaultValue = false lowerMultiplicity = 1 upperMultiplicity = 1

CanIfFixedBuffer: EcucBooleanParamDef

**Figure 10.3: AR\_EcucDef\_CanIfPrivateCfg**

##### CanIfPublicCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00246] |
| **Container Name** | CanIfPublicCfg |
| **Parent Container** | [CanIf](#_bookmark455) |
| **Description** | This container contains the public configuration (parameters) of the CAN Interface. |
| **Configuration Parameters** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfBusMirroringSupport [ECUC\_CanIf\_00847] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enable support for Bus Mirroring. | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |

|  |  |
| --- | --- |
| **Scope / Dependency** | scope: local |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDevErrorDetect [ECUC\_CanIf\_00614] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Switches the development error detection and notification on or off.   * true: detection and notification is enabled. * false: detection and notification is disabled. | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfEnableSecurityEventReporting [ECUC\_CanIf\_00848] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Switches the reporting of security events to the IdsM: - true: reporting is enabled. - false: reporting is disabled.  **Tags:**  atp.Status=draft | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfGlobalTimeSupport [ECUC\_CanIf\_00854] |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) |
| **Description** | Enables/Disables the Global Time APIs used when hardware timestamping is supported.  **Tags:**  atp.Status=draft |
| **Multiplicity** | 1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfMetaDataSupport [ECUC\_CanIf\_00824] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enable support for dynamic ID handling using L-SDU MetaData. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicCddHeaderFile [ECUC\_CanIf\_00671] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Defines header files for callback functions which shall be included in case of CDDs. Range of characters is 1.. 32. | | |
| **Multiplicity** | 0..\* | | |
| **Type**  **Default Value** | EcucStringParamDef | | |
|  | | |
| **Length** | 1–32 | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |

|  |  |
| --- | --- |
| **Scope / Dependency** | scope: ECU |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicHandleTypeEnum [ECUC\_CanIf\_00742] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | This parameter is used to configure the Can\_HwHandleType. The Can\_HwHandleType represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects the extended range shall be used (UINT16). | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | UINT16 |  | |
| UINT8 |  | |
| false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: Can\_HwHandleType | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicMultipleDrvSupport [ECUC\_CanIf\_00612] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Selects support for multiple CAN Drivers.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| true | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfPublicPnSupport [ECUC\_CanIf\_00772] |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) |
| **Description** | Selects support of Partial Network features in CanIf.  True: Enabled False: Disabled |
| **Multiplicity** | 1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
| false |
| **Post-Build Variant Value** | false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicReadRxPduDataApi [ECUC\_CanIf\_00607] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables / Disables the API CanIf\_ReadRxPduData() for reading received L-SDU data.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicReadRxPduNotifyStatusApi [ECUC\_CanIf\_00608] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables and disables the API for reading the notification status of receive L-PDUs.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfPublicReadTxPduNotifyStatusApi [ECUC\_CanIf\_00609] |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) |
| **Description** | Enables and disables the API for reading the notification status of transmit L-PDUs.  True: Enabled False: Disabled |
| **Multiplicity** | 1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
| false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicSetDynamicTxIdApi [ECUC\_CanIf\_00610] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables and disables the API for reconfiguration of the CAN Identifier for each Transmit L-PDU.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicTxBuffering [ECUC\_CanIf\_00618] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables and disables the buffering of transmit L-PDUs (rejected by the CanDrv) within the CAN Interface module.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfPublicTxConfirmPollingSupport [ECUC\_CanIf\_00733] |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) |
| **Description** | Configuration parameter to enable/disable the API to poll for Tx Confirmation state. |
| **Multiplicity** | 1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local  dependency: CAN State Manager module | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicWakeupCheckValidByNM [ECUC\_CanIf\_00741] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | If enabled, only NM messages shall validate a detected wake-up event in CanIf. If disabled, all received messages corresponding to a configured Rx PDU shall validate such a wake-up event. This parameter depends on CanIfPublicWakeupCheckValidSupport and shall only be configurable, if it is enabled.  True: Enabled False: Disabled | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfPublicWakeupCheckValidSupport | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfPublicWakeupCheckValidSupport [ECUC\_CanIf\_00611] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Selects support for wake up validation  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |

|  |  |
| --- | --- |
| **Scope / Dependency** | scope: ECU |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfSetBaudrateApi [ECUC\_CanIf\_00838] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Configuration parameter to enable/disable the CanIf\_SetBaudrate API to change the baud rate of a CAN Controller. If this parameter is set to true the CanIf\_SetBaudrate API shall be supported. Otherwise the API is not supported. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTriggerTransmitSupport [ECUC\_CanIf\_00844] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables the CanIf\_TriggerTransmit API at Pre-Compile-Time. Therefore, this parameter defines if there shall be support for trigger transmit transmissions. TRUE: Enabled FALSE: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| true | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfTxOfflineActiveSupport [ECUC\_CanIf\_00837] |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) |
| **Description** | Determines wether TxOffLineActive feature (see SWS\_CANIF\_00072) is supported by CanIf. True: Enabled False: Disabled |
| **Multiplicity** | 1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
| false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfVersionInfoApi [ECUC\_CanIf\_00613] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables and disables the API for reading the version information about the CAN Interface.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfWakeupSupport [ECUC\_CanIf\_00843] | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Enables the CanIf\_CheckWakeup API at Pre-Compile-Time. Therefore, this parameter defines if there shall be support for wake-up. TRUE: Enabled FALSE: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| true | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfSecurityEventRefs](#_bookmark560) | 0..1 | Container for the references to IdsMEvent elements representing the security events that the CanIf module shall report to the IdsM in case the coresponding security related event occurs (and if CanIfEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events.  **Tags:**  atp.Status=draft |



upperMultiplicity = 1 lowerMultiplicity = 1

defaultValue = false

+parameter

CanIfPublicReadRxPduNotifyStatusApi: EcucBooleanParamDef

+parameter CanIfPublicReadTxPduNotifyStatusApi:

EcucBooleanParamDef

defaultValue = False

defaultValue = False

CanIfPublicSetDynamicTxIdApi:

+parameter EcucBooleanParamDef

defaultValue = False

+parameter CanIfPublicWakeupCheckValidSupport:

EcucBooleanParamDef

defaultValue = False

+parameter

CanIfPublicMultipleDrvSupport: EcucBooleanParamDef

defaultValue = True

+parameter

CanIfVersionInfoApi:

EcucBooleanParamDef

defaultValue = false

CanIfDevErrorDetect: EcucBooleanParamDef

+parameter

defaultValue = false

+parameter CanIfPublicTxConfirmPollingSupport:

EcucBooleanParamDef

CanIfPublicTxBuffering:

+parameter EcucBooleanParamDef

+parameter CanIfTriggerTransmitSupport:

EcucBooleanParamDef

defaultValue = False

defaultValue = true

+parameter

CanIfTxOfflineActiveSupport:

EcucBooleanParamDef

defaultValue = False

CanIfSetBaudrateApi: EcucBooleanParamDef

+parameter

defaultValue = False lowerMultiplicity = 0 upperMultiplicity = 1

CanIfPublicWakeupCheckValidByNM: EcucBooleanParamDef

+parameter

defaultValue = false lowerMultiplicity = 0 upperMultiplicity = 1

+parameter CanIfBusMirroringSupport:

EcucBooleanParamDef

defaultValue = false

+parameter

+parameter

CanIfPublicPnSupport: EcucBooleanParamDef

CanIfPublicCddHeaderFile:

EcucStringParamDef

minLength = 1 maxLength = 32 lowerMultiplicity = 0 upperMultiplicity = \*

defaultValue = false

+parameter

+parameter

CanIfMetaDataSupport:

EcucBooleanParamDef

defaultValue = false lowerMultiplicity = 0 upperMultiplicity = 1

+literal

+parameter

CanIfPublicHandleTypeEnum: EcucEnumerationParamDef

UINT 8: EcucEnumerationLiteralDef

+literal

UINT16: EcucEnumerationLiteralDef

+parameter CanIfEnableSecurityEventReporting:

EcucBooleanParamDef

defaultValue = false

+parameter

CanIfGlobalTimeSupport: EcucBooleanParamDef

+subContainer

CanIfSecurityEventRefs: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfPublicCfg: EcucParamConfContainerDef

+parameter CanIfPublicReadRxPduDataApi:

EcucBooleanParamDef

defaultValue = true

CanIfWakeupSupport: EcucBooleanParamDef

**Figure 10.4: AR\_EcucDef\_CanIfPublicCfg**

##### CanIfInitCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00247] |
| **Container Name** | CanIfInitCfg |

|  |  |
| --- | --- |
| **Parent Container** | [CanIf](#_bookmark455) |
| **Description** | This container contains the init parameters of the CAN Interface. |
| **Configuration Parameters** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfInitCfgSet [ECUC\_CanIf\_00623] | | |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) | | |
| **Description** | Selects the CAN Interface specific configuration setup. This type of the external data structure shall contain the post build initialization data for the CAN Interface for all underlying CAN Dirvers.  constant to CanIf\_ConfigType | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucStringParamDef | | |
|  | | |
| **Length** | 1–32 | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfMaxBufferSize [ECUC\_CanIf\_00828] | | |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) | | |
| **Description** | Maximum total size of all Tx buffers. This parameter is needed only in case of post-build loadable implementation using static memory allocation. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 ..  18446744073709551615 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfMaxRxPduCfg [ECUC\_CanIf\_00830] | | |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) | | |
| **Description** | Maximum number of Pdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 ..  18446744073709551615 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfMaxTxPduCfg [ECUC\_CanIf\_00829] | | |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) | | |
| **Description** | Maximum number of Pdus. This parameter is needed only in case of post-build loadable implementation using static memory allocation. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 ..  18446744073709551615 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfBufferCfg](#_bookmark556) | 0..\* | 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. |
| [CanIfInitHohCfg](#_bookmark538) | 0..\* | This container contains the references to the configuration setup of each underlying CAN Driver. |
| [CanIfRxPduCfg](#_bookmark495) | 0..\* | This container contains the configuration (parameters) of each receive CAN L-PDU.  The SHORT-NAME of "CanIfRxPduConfig" container itself represents the symolic name of Receive L-PDU.  This L-SDU produces a meta data item of type CAN\_ID\_32. |
| [CanIfTxPduCfg](#_bookmark481) | 0..\* | This container contains the configuration (parameters) of a transmit CAN L-PDU. It has to be configured as often as a transmit CAN L-PDU is needed.  The SHORT-NAME of "CanIfTxPduConfig" container represents the symolic name of Transmit L-PDU.  This L-SDU consumes a meta data item of type CAN\_ID\_32. |



+container

+parameter

+parameter

+parameter

+parameter

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfMaxRxPduCfg: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfMaxTxPduCfg: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfMaxBufferSize: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfTxPduCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+subContainer

CanIfInitHohCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+subContainer

CanIfRxPduCfg: EcucParamConfContainerDef

+subContainer

minLength = 1 maxLength = 32

lowerMultiplicity = 1 upperMultiplicity = 1

CanIfInitCfgSet: EcucStringParamDef

CanIfInitCfg: EcucParamConfContainerDef

upperMultiplicity = 1 lowerMultiplicity = 0

CanIf: EcucModuleDef

**Figure 10.5: AR\_EcucDef\_CanIfInitCfg**

##### CanIfTxPduCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00248] |
| **Container Name** | CanIfTxPduCfg |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) |
| **Description** | This container contains the configuration (parameters) of a transmit CAN L-PDU. It has to be configured as often as a transmit CAN L-PDU is needed.  The SHORT-NAME of "CanIfTxPduConfig" container represents the symolic name of Transmit L-PDU.  This L-SDU consumes a meta data item of type CAN\_ID\_32. |
| **Post-Build Variant Multiplicity** | true |

|  |  |  |  |
| --- | --- | --- | --- |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduCanId [ECUC\_CanIf\_00592] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduCanIdMask [ECUC\_CanIf\_00823] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Identifier mask which denotes relevant bits in the CAN Identifier. This parameter may be used to keep parts of the CAN Identifier of dynamic transmit L-PDUs static. Range: 11 bits for Standard CAN Identifier, 29 bits for Extended CAN Identifier. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 3758096383 |  | |
| 3758096383 | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |

|  |  |  |  |
| --- | --- | --- | --- |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduCanIdType [ECUC\_CanIf\_00590] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Type of CAN Identifier of the transmit CAN L-PDU used by the CAN Driver module for CAN L-PDU transmission. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | 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) | |
| true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduId [ECUC\_CanIf\_00591] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | ECU wide unique, symbolic handle for transmit CAN L-SDU.  Range: 0..max. number of CantTxPduIds | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | |
| **Range**  **Default Value** | 0 .. 4294967295 |  | |
|  | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduPnFilterPdu [ECUC\_CanIf\_00773] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | If CanIfPublicPnFilterSupport is enabled, by this parameter PDUs could be configured which will pass the CanIfPnFilter.  If there is no CanIfTxPduPnFilterPdu configured per controller, the corresponding controller applies no CanIfPnFilter. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local  dependency: This parameter shall only be configurable if CanIfPublicPnSupport equals True. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduReadNotifyStatus [ECUC\_CanIf\_00589] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Enables and disables transmit confirmation for each transmit CAN L-SDU for reading its notification status.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local  dependency: CanIfPublicReadTxPduNotifyStatusApi must be enabled. | | |

|  |  |
| --- | --- |
| **Name** | CanIfTxPduTriggerTransmit [ECUC\_CanIf\_00840] |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) |
| **Description** | Determines if or if not CanIf shall use the trigger transmit API for this PDU. |
| **Multiplicity** | 0..1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
| false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU  dependency: If CanIfTxPduTriggerTransmit is TRUE then CanIfTxPduUserTxConfirmationUL has to be either PDUR or CDD and CanIfTxPduUserTriggerTransmitName has to be specified accordingly. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduTruncation [ECUC\_CanIf\_00845] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Enables/disables truncation of PDUs that exceed the configured size. | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| true | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduType [ECUC\_CanIf\_00593] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Defines the type of each transmit CAN L-PDU. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | DYNAMIC | CAN ID is defined at runtime. | |
| STATIC | CAN ID is defined at compile-time. | |
| true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduUserTriggerTransmitName [ECUC\_CanIf\_00842] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | This parameter defines the name of the <User\_TriggerTransmit>. 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\_TriggerTransmit> is fixed. If CanIfTxPduUserTxConfirmationUL equals CDD, the name of the <User\_TxConfirmation> is selectable.  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. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucFunctionNameDef | | |
|  | | |
| **Length** | 1–32 | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfTxPduUserTriggerTransmitName requires CanIfTxPduUserTxConfirmationUL to be either PDUR or CDD. | | |

|  |  |
| --- | --- |
| **Name** | CanIfTxPduUserTxConfirmationName [ECUC\_CanIf\_00528] |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) |
| **Description** | 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. |
| **Multiplicity** | 0..1 |
| **Type**  **Default Value** | EcucFunctionNameDef |
|  |
| **Length** | 1–32 |
| **Regular Expression** |  |
| **Post-Build Variant Multiplicity** | false |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduUserTxConfirmationUL [ECUC\_CanIf\_00527] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Multiplicity** | 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 | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduBufferRef [ECUC\_CanIf\_00831] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Configurable reference to a CanIf buffer configuration. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanIfBufferCfg | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTxPduRef [ECUC\_CanIf\_00603] | | |
| **Parent Container** | [CanIfTxPduCfg](#_bookmark481) | | |
| **Description** | Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to Pdu | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| CanIfTTTxFrame Triggering | 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. |



+parameter

+destination

+literal

+literal

+literal

+parameter

+literal

+literal

+literal

+literal

+literal

+parameter

+literal

+parameter

+parameter

+parameter

+parameter

+destination

+reference

+literal

+parameter

+literal

+parameter

+parameter

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1 maxLength = 32

defaultValue = false lowerMultiplicity = 0 upperMultiplicity = 1

CanIfTxPduUserTriggerTransmitName: EcucFunctionNameDef

CanIfTxPduTriggerTransmit: EcucBooleanParamDef

STATIC: EcucEnumerationLiteralDef

DYNAMIC: EcucEnumerationLiteralDef

CanIfTxPduType: EcucEnumerationParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfTxPduBufferRef: EcucReferenceDef

CanIfBufferCfg: EcucParamConfContainerDef

min = 0

max = 536870911

lowerMultiplicity = 0 upperMultiplicity = 1

defaultValue = false lowerMultiplicity = 0 upperMultiplicity = 1

CanIfTxPduCanId: EcucIntegerParamDef

+parameter

CanIfTxPduPnFilterPdu: EcucBooleanParamDef

min = 0

max = 3758096383

defaultValue = 3758096383 lowerMultiplicity = 0 upperMultiplicity = 1

defaultValue = true lowerMultiplicity = 1

upperMultiplicity = 1

CanIfTxPduTruncation: EcucBooleanParamDef

CanIfTxPduCanIdMask: EcucIntegerParamDef

+parameter

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1 maxLength = 32

symbolicNameValue = true min = 0

max = 4294967295

EcucFunctionNameDef

CanIfTxPduUserTxConfirmationName:

CanIfTxPduId: EcucIntegerParamDef

EXTENDED\_FD\_CAN: EcucEnumerationLiteralDef

+literal

STANDARD\_FD\_CAN: EcucEnumerationLiteralDef

EXTENDED\_CAN: EcucEnumerationLiteralDef

+literal

STANDARD\_CAN: EcucEnumerationLiteralDef

CanIfTxPduCanIdType: EcucEnumerationParamDef

CAN\_TSYN: EcucEnumerationLiteralDef

J1939NM: EcucEnumerationLiteralDef

CDD: EcucEnumerationLiteralDef

XCP: EcucEnumerationLiteralDef

J1939TP: EcucEnumerationLiteralDef

CAN\_NM: EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

PDUR: EcucEnumerationLiteralDef

CAN\_TP: EcucEnumerationLiteralDef

+literal

CanIfTxPduUserTxConfirmationUL: EcucEnumerationParamDef

+reference

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfTxPduRef: EcucReferenceDef

defaultValue = false

lowerMultiplicity = 0 upperMultiplicity = \*

Pdu: EcucParamConfContainerDef

CanIfTxPduReadNotifyStatus: EcucBooleanParamDef

CanIfTxPduCfg: EcucParamConfContainerDef

**Figure 10.6: AR\_EcucDef\_CanIfTxPduCfg**

##### CanIfRxPduCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00249] | | |
| **Container Name** | CanIfRxPduCfg | | |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) | | |
| **Description** | This container contains the configuration (parameters) of each receive CAN L-PDU.  The SHORT-NAME of "CanIfRxPduConfig" container itself represents the symolic name of Receive L-PDU.  This L-SDU produces a meta data item of type CAN\_ID\_32. | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduCanId [ECUC\_CanIf\_00598] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | CAN Identifier of Receive CAN L-PDUs used by the CAN Interface. Exa: Software Filtering. This parameter is used if exactly one Can Identifier is assigned to the Pdu. If a range is assigned then the CanIfRxPduCanIdRange parameter shall be used.  Range: 11 Bit For Standard CAN Identifier ... 29 Bit For Extended CAN identifier | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduCanIdMask [ECUC\_CanIf\_00822] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | Identifier mask which denotes relevant bits in the CAN Identifier. This parameter defines a CAN Identifier range in an alternative way to CanIfRxPduCanIdRange. It identifies the bits of the configured CAN Identifier that must match the received CAN Identifier. Range: 11 bits for Standard CAN Identifier, 29 bits for Extended CAN Identifier. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
| 536870911 | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduCanIdType [ECUC\_CanIf\_00596] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | CAN Identifier of receive CAN L-PDUs used by the CAN Driver for CAN L-PDU reception. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | 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\_CA  N | 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\_CA  N | CAN 2.0 frame with standard identifier  (11 bits) | |
| true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduDataLength [ECUC\_CanIf\_00599] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | Data length of the received CAN L-PDUs used by the CAN Interface. This information is used for Data Length Check. Additionally it might specify the valid bits in case of the discrete DLC for CAN FD L-PDUs > 8 bytes.  The data area size of a CAN L-PDU can have a range from 0 to 64 bytes. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 64 |  | |
|  | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU  dependency: If CanIfRxPduDataLength > 8 then CanIfRxPduCanIdType must not be STANDARD\_NO\_FD\_CAN or EXTENDED\_NO\_FD\_CAN | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduDataLengthCheck [ECUC\_CanIf\_00846] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | This parameter switches the message specific data length check. True: Data length check will be executed during the reception of this PDU. False: No data length check will be executed during the reception of this PDU. | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| true | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| **Name** | CanIfRxPduId [ECUC\_CanIf\_00597] | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | |
| **Description** | ECU wide unique, symbolic handle for receive CAN L-SDU. It shall fulfill ANSI/AUTOSAR definitions for constant defines.  Range: 0..max. number of defined CanRxPduIds | |
| **Multiplicity** | 1 | |
| **Type** | EcucIntegerParamDef (Symbolic Name generated for this parameter) | |
| **Range**  **Default Value** | 0 .. 4294967295 |  |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduReadData [ECUC\_CanIf\_00600] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | Enables and disables the Rx buffering for reading of received L-SDU data.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduReadNotifyStatus [ECUC\_CanIf\_00595] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | Enables and disables receive indication for each receive CAN L-SDU for reading its notification status.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local  dependency: CanIfPublicReadRxPduNotifyStatusApi must be enabled. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduUserRxIndicationName [ECUC\_CanIf\_00530] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucFunctionNameDef | | |
|  | | |
| **Length** | 1–32 | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| **Name** | CanIfRxPduUserRxIndicationUL [ECUC\_CanIf\_00529] | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | |
| **Description** | 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. | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range**  **Post-Build Variant Multiplicity** | 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 |
| false | |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduHrhIdRef [ECUC\_CanIf\_00602] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | The HRH to which Rx L-PDU belongs to, is referred through this parameter. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanIfHrhCfg | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local  dependency: This information has to be derived from the CAN Driver configuration. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduRef [ECUC\_CanIf\_00601] | | |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) | | |
| **Description** | Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to Pdu | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfRxPduCanIdRange](#_bookmark507) | 0..1 | Optional container that allows to map a range of CAN Ids to one PduId. |
| CanIfTTRxFrame Triggering | 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. |

lowerMultiplicity = 1 upperMultiplicity = 1

CanIfInitCfg: EcucParamConfContainerDef

+subContainer

CanIfRxPduCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+reference

+parameter

defaultValue = False

CanIfRxPduReadNotifyStatus: EcucBooleanParamDef

CanIfRxPduHrhIdRef: +de EcucReferenceDef

stination

CanIfHrhCfg:

EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+reference

requiresSymbolicNameValue = true

CanIfHrhIdSymRef: EcucReferenceDef

+parameter

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1

maxLength = 32

CanIfRxPduUserRxIndicationName: EcucFunctionNameDef

+reference

+destination

lowerMultiplicity = 0 upperMultiplicity = \*

Pdu: EcucParamConfContainerDef

CanIfRxPduRef: EcucReferenceDef

+literal

upperMultiplicity = 1 lowerMultiplicity = 0

CanIfRxPduUserRxIndicationUL: EcucEnumerationParamDef

CAN\_TP: EcucEnumerationLiteralDef

+literal

CDD: EcucEnumerationLiteralDef

+literal

+literal

CAN\_NM: EcucEnumerationLiteralDef

J1939TP: EcucEnumerationLiteralDef

+parameter

+literal

PDUR: EcucEnumerationLiteralDef



+literal

XCP: EcucEnumerationLiteralDef

+literal

J1939NM: EcucEnumerationLiteralDef

+literal

CAN\_TSYN: EcucEnumerationLiteralDef

+literal

EXTENDED\_CAN: EcucEnumerationLiteralDef

CanIfRxPduCanIdType: EcucEnumerationParamDef

+literal

STANDARD\_CAN: EcucEnumerationLiteralDef

+literal

+parameter

+literal

STANDARD\_FD\_CAN: EcucEnumerationLiteralDef

EXTENDED\_FD\_CAN: EcucEnumerationLiteralDef

+literal

EXTENDED\_NO\_FD\_CAN: EcucEnumerationLiteralDef

+literal

STANDARD\_NO\_FD\_CAN: EcucEnumerationLiteralDef

+parameter

CanIfRxPduDataLengthCheck: EcucBooleanParamDef

+parameter

symbolicNameValue = true upperMultiplicity = 1

lowerMultiplicity = 1 min = 0

max = 4294967295

CanIfRxPduId: EcucIntegerParamDef

defaultValue = true

+parameter

min = 0 max = 64

CanIfRxPduDataLength: EcucIntegerParamDef

+parameter

min = 0

max = 536870911

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfRxPduCanId: EcucIntegerParamDef

defaultValue = false

CanIfRxPduReadData: EcucBooleanParamDef

+parameter

CanIfRxPduCanIdRange:

E

CanIfRxPduCanIdRangeUpperCanId:

**Figure 10.7: AR\_EcucDef\_CanIfRxPduCfg**

##### CanIfRxPduCanIdRange

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00743] |
| **Container Name** | CanIfRxPduCanIdRange |
| **Parent Container** | [CanIfRxPduCfg](#_bookmark495) |
| **Description** | Optional container that allows to map a range of CAN Ids to one PduId. |
| **Configuration Parameters** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduCanIdRangeLowerCanId [ECUC\_CanIf\_00745] | | |
| **Parent Container** | [CanIfRxPduCanIdRange](#_bookmark507) | | |
| **Description** | Lower CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids are mapped to one PduId. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfRxPduCanIdRangeUpperCanId [ECUC\_CanIf\_00744] | | |
| **Parent Container** | [CanIfRxPduCanIdRange](#_bookmark507) | | |
| **Description** | Upper CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids are mapped to one PduId. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

**No Included Containers**

##### CanIfDispatchCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00250] |
| **Container Name** | CanIfDispatchCfg |
| **Parent Container** | [CanIf](#_bookmark455) |

|  |  |
| --- | --- |
| **Description** | Callback functions provided by upper layer modules of the CanIf. The callback functions defined in this container are common to all configured CAN Driver / CAN Transceiver Driver modules. |
| **Configuration Parameters** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserCheckTrcvWakeFlagIndicationName [ECUC\_CanIf\_00791] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucFunctionNameDef | | |
|  | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfDispatchUserCheckTrcvWakeFlagIndicationUL, CanIfPublicPnSupport | | |

|  |  |  |
| --- | --- | --- |
| **Name** | CanIfDispatchUserCheckTrcvWakeFlagIndicationUL [ECUC\_CanIf\_00792] | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | |
| **Description** | This parameter defines the upper layer module to which the CheckTrcvWakeFlagIndication from the Driver modules have to be routed. If CanIfPublicPnSupport equals False, this parameter shall not be configurable. | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucEnumerationParamDef | |
| **Range**  **Post-Build Variant Multiplicity** | CAN\_SM | CAN State Manager |
| CDD | Complex Driver |
| false | |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserClearTrcvWufFlagIndicationName [ECUC\_CanIf\_00789] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucFunctionNameDef | | |
|  | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfDispatchUserClearTrcvWufFlagIndicationUL, CanIfPublicPnSupport | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserClearTrcvWufFlagIndicationUL [ECUC\_CanIf\_00790] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | This parameter defines the upper layer module to which the ClearTrcvWufFlagIndication from the Driver modules have to be routed. If CanIfPublicPnSupport equals False, this parameter shall not be configurable. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Multiplicity** | CAN\_SM | CAN State Manager | |
| CDD | Complex Driver | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfPublicPnSupport | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserConfirmPnAvailabilityName [ECUC\_CanIf\_00819] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | This parameter defines the name of <User\_ConfirmPnAvailability>. If CanIfDispatchUserConfirmPnAvailabilityUL equals CAN\_SM the name of <User\_ConfirmPnAvailability> is fixed. If it equals CDD, the name is selectable. If CanIfPublicPnSupport equals False, this parameter shall not be configurable. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucFunctionNameDef | | |
|  | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfDispatchUserConfirmPnAvailabilityUL, CanIfPublicPnSupport | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserConfirmPnAvailabilityUL [ECUC\_CanIf\_00820] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Multiplicity** | CAN\_SM | CAN State Manager | |
| CDD | Complex Driver | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfPublicPnSupport | | |

|  |  |
| --- | --- |
| **Name** | CanIfDispatchUserCtrlBusOffName [ECUC\_CanIf\_00525] |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) |
| **Description** | This parameter defines the name of <User\_ControllerBusOff>. This parameter depends on the parameter CanIfDispatchUserCtrlBusOffUL. If CanIfDispatchUserCtrlBusOffUL equals CAN\_SM the name of  <User\_ControllerBusOff> is fixed. If CanIfDispatchUserCtrlBusOffUL  equals CDD, the name of <User\_ControllerBusOff> is selectable. |
| **Multiplicity** | 0..1 |
| **Type**  **Default Value** | EcucFunctionNameDef |
|  |
| **Length** | 1–32 |
| **Regular Expression** |  |
| **Post-Build Variant Multiplicity** | false |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserCtrlBusOffUL [ECUC\_CanIf\_00547] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | This parameter defines the upper layer (UL) module to which the notifications of all ControllerBusOff events from the CAN Driver modules have to be routed via <User\_ControllerBusOff>. There is no possibility to configure no upper layer (UL) module as the provider of  <User\_ControllerBusOff>. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | CAN\_SM | CAN State Manager | |
| CDD | Complex Driver | |
| false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfDispatchUserCtrlModeIndicationName [ECUC\_CanIf\_00683] |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) |
| **Description** | This parameter defines the name of <User\_ControllerModeIndication>. This parameter depends on the parameter CanIfDispatchUserCtrlModeIndicationUL. If CanIfDispatchUserCtrlModeIndicationUL equals CAN\_SM the name of  <User\_ControllerModeIndication> is fixed. If CanIfDispatchUserCtrlModeIndicationUL equals CDD, the name of  <User\_ControllerModeIndication> is selectable. |
| **Multiplicity** | 0..1 |
| **Type**  **Default Value** | EcucFunctionNameDef |
|  |
| **Length** | 1–32 |
| **Regular Expression** |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserCtrlModeIndicationUL [ECUC\_CanIf\_00684] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | This parameter defines the upper layer (UL) module to which the notifications of all ControllerTransition events from the CAN Driver modules have to be routed via <User\_ControllerModeIndication>. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | CAN\_SM | CAN State Manager | |
| CDD | Complex Driver | |
| false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfDispatchUserTrcvModeIndicationName [ECUC\_CanIf\_00685] |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) |
| **Description** | This parameter defines the name of <User\_TrcvModeIndication>. This parameter depends on the parameter CanIfDispatchUserTrcvModeIndicationUL. If CanIfDispatchUserTrcvModeIndicationUL equals CAN\_SM the name of <User\_TrcvModeIndication> is fixed. If CanIfDispatchUserTrcvModeIndicationUL equals CDD, the name of  <User\_TrcvModeIndication> is selectable. |
| **Multiplicity** | 0..1 |
| **Type**  **Default Value** | EcucFunctionNameDef |
|  |
| **Length** | 1–32 |
| **Regular Expression** |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserTrcvModeIndicationUL [ECUC\_CanIf\_00686] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | This parameter defines the upper layer (UL) module to which the notifications of all TransceiverTransition events from the CAN Transceiver Driver modules have to be routed via  <User\_TrcvModeIndication>. If no UL module is configured, no upper  layer callback function will be called. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Multiplicity** | CAN\_SM | CAN State Manager | |
| CDD | Complex Driver | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserValidateWakeupEventName [ECUC\_CanIf\_00531] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | 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. | | |
| **Multiplicity** | 0..1 | | |
| **Type**  **Default Value** | EcucFunctionNameDef | | |
|  | | |
| **Length** | 1–32 | | |
| **Regular Expression** |  | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: CanIfDispatchUserValidateWakeupEventUL | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfDispatchUserValidateWakeupEventUL [ECUC\_CanIf\_00549] | | |
| **Parent Container** | [CanIfDispatchCfg](#_bookmark509) | | |
| **Description** | This parameter defines the upper layer (UL) module to which the notifications about positive former requested wake up sources have to be routed via <User\_ValidateWakeupEvent>. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Multiplicity** | CDD | Complex Driver | |
| ECUM | ECU State Manager | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

**No Included Containers**



+parameter

+parameter

+literal

+parameter

+literal

+parameter

+literal

+literal

+parameter

+literal

+parameter

+literal

+literal

+literal

+parameter

+literal

+parameter

+literal

+literal

+parameter

+literal

+literal

CDD: EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

CAN\_SM: EcucEnumerationLiteralDef

+literal

CanIfDispatchUserConfirmPnAvailabilityUL: EcucEnumerationParamDef

CDD: EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

+parameter

CAN\_SM: EcucEnumerationLiteralDef

CanIfDispatchUserCheckTrcvWakeFlagIndicationUL: EcucEnumerationParamDef

CDD: EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

CAN\_SM: EcucEnumerationLiteralDef

CanIfDispatchUserClearTrcvWufFlagIndicationUL: EcucEnumerationParamDef

lowerMultiplicity = 0 upperMultiplicity = 1

+parameter

CanIfDispatchUserCheckTrcvWakeFlagIndicationName: EcucFunctionNameDef

lowerMultiplicity = 0 upperMultiplicity = 1

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfDispatchUserClearTrcvWufFlagIndicationName: EcucFunctionNameDef

+parameter

CanIfDispatchUserConfirmPnAvailabilityName: EcucFunctionNameDef

CDD: EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

CAN\_SM: EcucEnumerationLiteralDef

CanIfDispatchUserTrcvModeIndicationUL: EcucEnumerationParamDef

CDD: EcucEnumerationLiteralDef

lowerMultiplicity = 1 upperMultiplicity = 1

+parameter

CAN\_SM: EcucEnumerationLiteralDef

CanIfDispatchUserCtrlModeIndicationUL: EcucEnumerationParamDef

+parameter

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1 maxLength = 32

CanIfDispatchUserTrcvModeIndicationName: EcucFunctionNameDef

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1

maxLength = 32

CanIfDispatchUserCtrlModeIndicationName: EcucFunctionNameDef

CDD: EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

ECUM: EcucEnumerationLiteralDef

CanIfDispatchUserValidateWakeupEventUL: EcucEnumerationParamDef

CDD: EcucEnumerationLiteralDef

CAN\_SM: EcucEnumerationLiteralDef

CanIfDispatchUserCtrlBusOffUL: EcucEnumerationParamDef

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1 maxLength = 32

CanIfDispatchUserValidateWakeupEventName: EcucFunctionNameDef

lowerMultiplicity = 0 upperMultiplicity = 1 minLength = 1

maxLength = 32

CanIfDispatchCfg: EcucParamConfContainerDef

CanIfDispatchUserCtrlBusOffName: EcucFunctionNameDef

**Figure 10.8: AR\_EcucDef\_CanIfDispatchCfg**

##### CanIfCtrlCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00546] |

|  |  |  |  |
| --- | --- | --- | --- |
| **Container Name** | CanIfCtrlCfg | | |
| **Parent Container** | [CanIfCtrlDrvCfg](#_bookmark530) | | |
| **Description** | This container contains the configuration (parameters) of an adressed CAN controller by an underlying CAN Driver module. This container is configurable per CAN controller. | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD |
| – |  |
| – |  |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfCtrlId [ECUC\_CanIf\_00647] | | |
| **Parent Container** | [CanIfCtrlCfg](#_bookmark525) | | |
| **Description** | This parameter abstracts from the CAN Driver specific parameter Controller. Each controller of all connected CAN Driver modules shall be assigned to one specific ControllerId of the CanIf. Range: 0..number of configured controllers of all CAN Driver modules | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | |
| **Range**  **Default Value** | 0 .. 255 |  | |
|  | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfCtrlWakeupSupport [ECUC\_CanIf\_00637] | | |
| **Parent Container** | [CanIfCtrlCfg](#_bookmark525) | | |
| **Description** | This parameter defines if a respective controller of the referenced CAN Driver modules is queriable for wake up events.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfCtrlCanCtrlRef [ECUC\_CanIf\_00636] | | |
| **Parent Container** | [CanIfCtrlCfg](#_bookmark525) | | |
| **Description** | This parameter references to the logical handle of the underlying CAN controller from the CAN Driver module to be served by the CAN Interface module. The following parameters of CanController config container shall be referenced by this link: CanControllerId, CanWakeupSourceRef  Range: 0..max. number of underlying supported CAN controllers | | |
| **Multiplicity** | 1 | | |
| **Type** | Symbolic name reference to CanController | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: amount of CAN controllers | | |

**No Included Containers**



+container

+subContainer

+reference

+destination

+parameter

+parameter

defaultValue = False

CanIfCtrlWakeupSupport: EcucBooleanParamDef

upperMultiplicity = 1 lowerMultiplicity = 1

symbolicNameValue = true min = 0

max = 255

CanControllerId: EcucIntegerParamDef

min = 0 max = 255

symbolicNameValue = true

+parameter

CanIfCtrlId: EcucIntegerParamDef

upperMultiplicity = \* lowerMultiplicity = 1

requiresSymbolicNameValue = true

upperMultiplicity = \* lowerMultiplicity = 1

CanController: EcucParamConfContainerDef

CanIfCtrlCanCtrlRef: EcucReferenceDef

CanIfCtrlCfg: EcucParamConfContainerDef

lowerMultiplicity = 1 upperMultiplicity = \*

CanIfCtrlDrvCfg: EcucParamConfContainerDef

upperMultiplicity = 1 lowerMultiplicity = 0

CanIf: EcucModuleDef

**Figure 10.9: AR\_EcucDef\_CanIfCtrlCfg**

##### CanIfCtrlDrvCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00253] | | |
| **Container Name** | CanIfCtrlDrvCfg | | |
| **Parent Container** | [CanIf](#_bookmark455) | | |
| **Description** | Configuration parameters for all the underlying CAN Driver modules are aggregated under this container. For each CAN Driver module a seperate instance of this container has to be provided. | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD |
| – |  |
| – |  |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfCtrlDrvInitHohConfigRef [ECUC\_CanIf\_00642] | | |
| **Parent Container** | [CanIfCtrlDrvCfg](#_bookmark530) | | |
| **Description** | Reference to the Init Hoh Configuration | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanIfInitHohCfg | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfCtrlDrvNameRef [ECUC\_CanIf\_00638] | | |
| **Parent Container** | [CanIfCtrlDrvCfg](#_bookmark530) | | |
| **Description** | CAN Interface Driver Reference.  This reference can be used to get any information (Ex. Driver Name, Vendor ID) from the CAN driver.  The CAN Driver name can be derived from the ShortName of the CAN driver module. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanGeneral | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfCtrlCfg](#_bookmark525) | 1..\* | This container contains the configuration (parameters) of an adressed CAN controller by an underlying CAN Driver module. This container is configurable per CAN controller. |



upperMultiplicity = 1 lowerMultiplicity = 0

CanIf: EcucModuleDef

+destination

upperMultiplicity = 1 lowerMultiplicity = 1

CanIfCtrlDrvNameRef: EcucReferenceDef

CanGeneral: EcucParamConfContainerDef

+container

+reference

+reference

lowerMultiplicity = 1 upperMultiplicity = \*

CanIfCtrlDrvCfg: EcucParamConfContainerDef

+destination

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfCtrlDrvInitHohConfigRef: EcucReferenceDef

CanIfInitHohCfg: EcucParamConfContainerDef



**Figure 10.10: AR\_EcucDef\_CanIfCtrlDrvCfg**

##### CanIfTrcvDrvCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00273] | | |
| **Container Name** | CanIfTrcvDrvCfg | | |
| **Parent Container** | [CanIf](#_bookmark455) | | |
| **Description** | This container contains the configuration (parameters) of all addressed CAN transceivers by each underlying CAN Transceiver Driver module. For each CAN transceiver Driver a seperate instance of this container shall be provided. | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD |
| – |  |
| – |  |
| **Configuration Parameters** | | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfTrcvCfg](#_bookmark534) | 1..\* | This container contains the configuration (parameters) of one addressed CAN transceiver by the underlying CAN Transceiver Driver module. For each CAN transceiver a seperate instance of this container has to be provided. |



+container

+subContainer

lowerMultiplicity = 1 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfTrcvCfg: EcucParamConfContainerDef

CanIfTrcvDrvCfg: EcucParamConfContainerDef

upperMultiplicity = 1 lowerMultiplicity = 0

CanIf: EcucModuleDef

**Figure 10.11: AR\_EcucDef\_CanIfTrcvDrvCfg**

##### CanIfTrcvCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00587] | | |
| **Container Name** | CanIfTrcvCfg | | |
| **Parent Container** | [CanIfTrcvDrvCfg](#_bookmark532) | | |
| **Description** | This container contains the configuration (parameters) of one addressed CAN transceiver by the underlying CAN Transceiver Driver module. For each CAN transceiver a seperate instance of this container has to be provided. | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD |
| – |  |
| – |  |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTrcvId [ECUC\_CanIf\_00654] | | |
| **Parent Container** | [CanIfTrcvCfg](#_bookmark534) | | |
| **Description** | This parameter abstracts from the CAN Transceiver Driver specific parameter Transceiver. Each transceiver of all connected CAN Transceiver Driver modules shall be assigned to one specific TransceiverId of the CanIf.  Range: 0..number of configured transceivers of all CAN Transceiver Driver modules | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | |
| **Range**  **Default Value** | 0 .. 255 |  | |
|  | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfTrcvWakeupSupport [ECUC\_CanIf\_00606] | | |
| **Parent Container** | [CanIfTrcvCfg](#_bookmark534) | | |
| **Description** | This parameter defines if a respective transceiver of the referenced CAN Transceiver Driver modules is queriable for wake up events.  True: Enabled False: Disabled | | |
| **Multiplicity** | 1 | | |
| **Type**  **Default Value** | EcucBooleanParamDef | | |
| false | | |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU | | |

|  |  |
| --- | --- |
| **Name** | CanIfTrcvCanTrcvRef [ECUC\_CanIf\_00605] |
| **Parent Container** | [CanIfTrcvCfg](#_bookmark534) |
| **Description** | This parameter references to the logical handle of the underlying CAN transceiver from the CAN transceiver driver module to be served by the CAN Interface module.  Range: 0..max. number of underlying supported CAN transceivers |
| **Multiplicity** | 1 |
| **Type** | Symbolic name reference to CanTrcvChannel |
| **Post-Build Variant Value** | false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: ECU  dependency: amount of CAN transceivers | | |

**No Included Containers**



+container

+subContainer

+parameter

+parameter

+reference

+destination

+reference

symbolicNameValue = true max = 255

lowerMultiplicity = 0 upperMultiplicity = 1

requiresSymbolicNameValue = true

CanTrcvChannelId: EcucIntegerParamDef

CanTrcvWakeupSourceRef: EcucReferenceDef

+parameter

upperMultiplicity = \* lowerMultiplicity = 1

lowerMultiplicity = 1 upperMultiplicity = 1

requiresSymbolicNameValue = true

CanTrcvChannel: EcucParamConfContainerDef

CanIfTrcvCanTrcvRef: EcucReferenceDef

min = 0 max = 255

symbolicNameValue = true

CanIfTrcvId: EcucIntegerParamDef

defaultValue = false

lowerMultiplicity = 1 upperMultiplicity = \*

CanIfTrcvWakeupSupport: EcucBooleanParamDef

CanIfTrcvCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfTrcvDrvCfg: EcucParamConfContainerDef

upperMultiplicity = 1 lowerMultiplicity = 0

CanIf: EcucModuleDef

**Figure 10.12: AR\_EcucDef\_CanIfTrcvCfg**

##### CanIfInitHohCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00257] |
| **Container Name** | CanIfInitHohCfg |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) |
| **Description** | This container contains the references to the configuration setup of each underlying CAN Driver. |
| **Post-Build Variant Multiplicity** | false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD |
| – |  |
| – |  |
| **Configuration Parameters** | | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfHrhCfg](#_bookmark544) | 0..\* | This container contains configuration parameters for each hardware receive object (HRH). |
| [CanIfHthCfg](#_bookmark540) | 0..\* | This container contains parameters related to each HTH. |



+subContainer

+subContainer

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfHthCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

+subContainer

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfHrhCfg: EcucParamConfContainerDef

CanIfInitHohCfg: EcucParamConfContainerDef

lowerMultiplicity = 1 upperMultiplicity = 1

CanIfInitCfg: EcucParamConfContainerDef

**Figure 10.13: AR\_EcucDef\_CanIfInitHohCfg**

##### CanIfHthCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00258] | | |
| **Container Name** | CanIfHthCfg | | |
| **Parent Container** | [CanIfInitHohCfg](#_bookmark538) | | |
| **Description** | This container contains parameters related to each HTH. | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHthCanCtrlIdRef [ECUC\_CanIf\_00625] | | |
| **Parent Container** | [CanIfHthCfg](#_bookmark540) | | |
| **Description** | Reference to controller Id to which the HTH belongs to. A controller can contain one or more HTHs. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanIfCtrlCfg | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHthIdSymRef [ECUC\_CanIf\_00627] | | |
| **Parent Container** | [CanIfHthCfg](#_bookmark540) | | |
| **Description** | The parameter refers to a particular HTH object in the CanDrv configuration (see CanHardwareObject ECUC\_Can\_00324). CanIf receives the following information of the CanDrv module by this reference:   * CanHandleType (see ECUC\_Can\_00323) * CanObjectId (see ECUC\_Can\_00326) | | |
| **Multiplicity** | 1 | | |
| **Type** | Symbolic name reference to CanHardwareObject | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

**No Included Containers**



+subContainer

+reference

+destination

+reference

+destination

+parameter

upperMultiplicity = 1 lowerMultiplicity = 1

symbolicNameValue = true min = 0

max = 65535

CanHandleType: EcucEnumerationParamDef

CanObjectId: EcucIntegerParamDef

+parameter

upperMultiplicity = \* lowerMultiplicity = 1

requiresSymbolicNameValue = true

CanHardwareObject: EcucParamConfContainerDef

CanIfHthIdSymRef: EcucReferenceDef

lowerMultiplicity = 0 upperMultiplicity = \*

upperMultiplicity = \* lowerMultiplicity = 1

CanIfHthCanCtrlIdRef: EcucReferenceDef

CanIfCtrlCfg: EcucParamConfContainerDef

CanIfHthCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfInitHohCfg: EcucParamConfContainerDef

**Figure 10.14: AR\_EcucDef\_CanIfHthCfg**

##### CanIfHrhCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00259] | | |
| **Container Name** | CanIfHrhCfg | | |
| **Parent Container** | [CanIfInitHohCfg](#_bookmark538) | | |
| **Description** | This container contains configuration parameters for each hardware receive object (HRH). | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Configuration Parameters** | | | |

|  |  |
| --- | --- |
| **Name** | CanIfHrhSoftwareFilter [ECUC\_CanIf\_00632] |
| **Parent Container** | [CanIfHrhCfg](#_bookmark544) |
| **Description** | Selects the hardware receive objects by using the HRH range/list from CAN Driver configuration to define, for which HRH a software filtering has to be performed at during receive processing.  True: Software filtering is enabled False: Software filtering is enabled |
| **Multiplicity** | 1 |
| **Type**  **Default Value** | EcucBooleanParamDef |
| true |

|  |  |  |  |
| --- | --- | --- | --- |
| **Post-Build Variant Value** | false | | |
| **Value Configuration Class** | **Pre-compile time Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME,  VARIANT-POST-BUILD |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHrhCanCtrlIdRef [ECUC\_CanIf\_00631] | | |
| **Parent Container** | [CanIfHrhCfg](#_bookmark544) | | |
| **Description** | Reference to controller Id to which the HRH belongs to. A controller can contain one or more HRHs. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanIfCtrlCfg | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHrhIdSymRef [ECUC\_CanIf\_00634] | | |
| **Parent Container** | [CanIfHrhCfg](#_bookmark544) | | |
| **Description** | The parameter refers to a particular HRH object in the CanDrv configuration (see CanHardwareObject ECUC\_Can\_00324). CanIf receives the following information of the CanDrv module by this reference:   * CanHandleType (see ECUC\_Can\_00323) * CanObjectId (see ECUC\_Can\_00326) | | |
| **Multiplicity** | 1 | | |
| **Type** | Symbolic name reference to CanHardwareObject | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| **Included Containers** | | |
| **Container Name** | **Multiplicity** | **Scope / Dependency** |
| [CanIfHrhRangeCfg](#_bookmark549) | 0..\* | Defines the parameters required for configurating multiple CANID ranges for a given same HRH. |



+subContainer

+parameter

upperMultiplicity = 1 lowerMultiplicity = 1

symbolicNameValue = true min = 0

max = 65535

CanObjectId: EcucIntegerParamDef

CanHandleType: EcucEnumerationParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

+parameter

CanIfHrhRangeCfg: EcucParamConfContainerDef

defaultValue = True

CanIfHrhSoftwareFilter: EcucBooleanParamDef

upperMultiplicity = \* lowerMultiplicity = 1

requiresSymbolicNameValue = true

+subContainer

CanHardwareObject: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfHrhCfg: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfInitHohCfg: EcucParamConfContainerDef

+reference

CanIfHrhIdSymRef: EcucReferenceDef

+reference

+destination

+parameter

upperMultiplicity = \* lowerMultiplicity = 1

CanIfHrhCanCtrlIdRef: EcucReferenceDef

CanIfCtrlCfg: EcucParamConfContainerDef

+destination

**Figure 10.15: AR\_EcucDef\_CanIfHrhCfg**

##### CanIfHrhRangeCfg

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00628] | | |
| **Container Name** | CanIfHrhRangeCfg | | |
| **Parent Container** | [CanIfHrhCfg](#_bookmark544) | | |
| **Description** | Defines the parameters required for configurating multiple CANID ranges for a given same HRH. | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHrhRangeBaseId [ECUC\_CanIf\_00825] | | |
| **Parent Container** | [CanIfHrhRangeCfg](#_bookmark549) | | |
| **Description** | CAN Identifier used as base value in combination with CanIfHrhRangeMask for a masked ID range in which all CAN Ids shall pass the software filtering. The size of this parameter is limited by CanIfHrhRangeRxPduRangeCanIdType. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHrhRangeMask [ECUC\_CanIf\_00826] | | |
| **Parent Container** | [CanIfHrhRangeCfg](#_bookmark549) | | |
| **Description** | Used as mask value in combination with CanIfHrhRangeBaseId for a masked ID range in which all CAN Ids shall pass the software filtering. The size of this parameter is limited by CanIfHrhRangeRxPduRangeCanIdType. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHrhRangeRxPduLowerCanId [ECUC\_CanIf\_00629] | | |
| **Parent Container** | [CanIfHrhRangeCfg](#_bookmark549) | | |
| **Description** | Lower CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids shall pass the software filtering. | | |
| **Multiplicity** | 0..1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 536870911 |  | |
|  | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Post-Build Variant Value** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfHrhRangeRxPduRangeCanIdType [ECUC\_CanIf\_00644] | | |
| **Parent Container** | [CanIfHrhRangeCfg](#_bookmark549) | | |
| **Description** | Specifies whether a configured Range of CAN Ids shall only consider standard CAN Ids or extended CAN Ids. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucEnumerationParamDef | | |
| **Range**  **Post-Build Variant Value** | EXTENDED | All the CANIDs are of type extended only (29 bit). | |
| STANDARD | All the CANIDs are of type standard  only (11bit). | |
| true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| **Name** | CanIfHrhRangeRxPduUpperCanId [ECUC\_CanIf\_00630] | |
| **Parent Container** | [CanIfHrhRangeCfg](#_bookmark549) | |
| **Description** | Upper CAN Identifier of a receive CAN L-PDU for identifier range definition, in which all CAN Ids shall pass the software filtering. | |
| **Multiplicity** | 0..1 | |
| **Type** | EcucIntegerParamDef | |
| **Range**  **Default Value** | 0 .. 536870911 |  |
|  | |
| **Post-Build Variant Multiplicity** | true | |

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

**No Included Containers**



+subContainer

+parameter

+literal

+literal

upperMultiplicity = 1 lowerMultiplicity = 0 min = 0

max = 536870911

CanIfHrhRangeMask: EcucIntegerParamDef

+parameter

upperMultiplicity = 1 lowerMultiplicity = 0 min = 0

max = 536870911

CanIfHrhRangeBaseId: EcucIntegerParamDef

+parameter

EXTENDED:

EcucEnumerationLiteralDef

CanIfHrhRangeRxPduRangeCanIdType: EcucEnumerationParamDef

STANDARD:

EcucEnumerationLiteralDef

+parameter

upperMultiplicity = 1 lowerMultiplicity = 0 min = 0

max = 536870911

CanIfHrhRangeRxPduLowerCanId: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

+parameter

min = 0

max = 536870911

lowerMultiplicity = 0 upperMultiplicity = 1

CanIfHrhRangeCfg: EcucParamConfContainerDef

CanIfHrhRangeRxPduUpperCanId: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfHrhCfg: EcucParamConfContainerDef

**Figure 10.16: AR\_EcucDef\_CanIfHrhRangeCfg**

##### CanIfBufferCfg

|  |  |
| --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00832] |
| **Container Name** | CanIfBufferCfg |
| **Parent Container** | [CanIfInitCfg](#_bookmark479) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | This container contains the Txbuffer configuration. Multiple buffers with different sizes could be configured. If CanIfBufferSize (ECUC\_CanIf\_00834) equals 0, the CanIf Tx L-PDU only refers via this CanIfBufferCfg the corresponding CanIfHthCfg. | | |
| **Post-Build Variant Multiplicity** | true | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Configuration Parameters** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfBufferSize [ECUC\_CanIf\_00834] | | |
| **Parent Container** | [CanIfBufferCfg](#_bookmark556) | | |
| **Description** | This parameter defines the number of CanIf Tx L-PDUs which can be buffered in one Txbuffer. If this value equals 0, the CanIf does not perform Txbuffering for the CanIf Tx L-PDUs which are assigned to this Txbuffer. If CanIfPublicTxBuffering equals False, this parameter equals 0 for all TxBuffer. If the CanHandleType of the referred HTH equals FULL, this parameter equals 0 for this TxBuffer. | | |
| **Multiplicity** | 1 | | |
| **Type** | EcucIntegerParamDef | | |
| **Range**  **Default Value** | 0 .. 255 |  | |
| 0 | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |
| **Scope / Dependency** | scope: local  dependency: CanIfPublicTxBuffering, CanHandleType | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CanIfBufferHthRef [ECUC\_CanIf\_00833] | | |
| **Parent Container** | [CanIfBufferCfg](#_bookmark556) | | |
| **Description** | Reference to HTH, that defines the hardware object or the pool of hardware objects configured for transmission. All the CanIf Tx L-PDUs refer via the CanIfBufferCfg and this parameter to the HTHs if TxBuffering is enabled, or not.  Each HTH shall not be assigned to more than one buffer. | | |
| **Multiplicity** | 1 | | |
| **Type** | Reference to CanIfHthCfg | | |
| **Post-Build Variant Value** | true | | |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | VARIANT-PRE-COMPILE |
| X | VARIANT-LINK-TIME |
| X | VARIANT-POST-BUILD |

|  |  |
| --- | --- |
| **Scope / Dependency** | scope: local |

**No Included Containers**



+reference

+parameter

min = 0 max = 255

defaultValue = 0

CanIfBufferSize: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfBufferCfg: EcucParamConfContainerDef

+destination

lowerMultiplicity = 0 upperMultiplicity = \*

CanIfHthCfg: EcucParamConfContainerDef

CanIfBufferHthRef: EcucReferenceDef

**Figure 10.17: AR\_EcucDef\_CanIfBufferCfg**

##### CanIfSecurityEventRefs

|  |  |  |  |
| --- | --- | --- | --- |
| **SWS Item** | [ECUC\_CanIf\_00849] | | |
| **Container Name** | CanIfSecurityEventRefs | | |
| **Parent Container** | [CanIfPublicCfg](#_bookmark461) | | |
| **Description** | Container for the references to IdsMEvent elements representing the security events that the CanIf module shall report to the IdsM in case the coresponding security related event occurs (and if CanIfEnableSecurityEventReporting is set to "true"). The standardized security events in this container can be extended by vendor-specific security events.  **Tags:**  atp.Status=draft | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Configuration Parameters** | | | |

|  |  |
| --- | --- |
| **Name** | CANIF\_SEV\_ERRORSTATE\_BUSOFF [ECUC\_CanIf\_00853] |
| **Parent Container** | [CanIfSecurityEventRefs](#_bookmark560) |
| **Description** | The CAN controller transitioned to state busoff.  **Tags:**  atp.Status=draft |
| **Multiplicity** | 0..1 |
| **Type** | Symbolic name reference to IdsMEvent |
| **Post-Build Variant Multiplicity** | false |
| **Post-Build Variant Value** | false |

|  |  |  |  |
| --- | --- | --- | --- |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CANIF\_SEV\_ERRORSTATE\_PASSIVE [ECUC\_CanIf\_00852] | | |
| **Parent Container** | [CanIfSecurityEventRefs](#_bookmark560) | | |
| **Description** | A reception related error was detected. Depending on the context data this could indicate suspicious CAN activity.  **Tags:**  atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | Symbolic name reference to IdsMEvent | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | CANIF\_SEV\_TX\_ERROR\_DETECTED [ECUC\_CanIf\_00850] | | |
| **Parent Container** | [CanIfSecurityEventRefs](#_bookmark560) | | |
| **Description** | A transmission related error was detected. Depending on the context data this could indicate suspicious CAN activity.  **Tags:**  atp.Status=draft | | |
| **Multiplicity** | 0..1 | | |
| **Type** | Symbolic name reference to IdsMEvent | | |
| **Post-Build Variant Multiplicity** | false | | |
| **Post-Build Variant Value** | false | | |
| **Multiplicity Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Value Configuration Class** | **Pre-compile time**  **Link time**  **Post-build time** | X | All Variants |
| – |  |
| – |  |
| **Scope / Dependency** | scope: local | | |

**No Included Containers**



CanIfPublicCfg: EcucParamConfContainerDef

+parameter CanIfEnableSecurityEventReporting:

EcucBooleanParamDef

upperMultiplicity = 1 lowerMultiplicity = 1

defaultValue = false

+subContainer

CanIfSecurityEventRefs: EcucParamConfContainerDef

+reference

CANIF\_SEV\_TX\_ERROR\_DETECTED:

EcucReferenceDef

+destination

IdsMEvent: EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = 1

lowerMultiplicity = 0 upperMultiplicity = 1

requiresSymbolicNameValue = true

lowerMultiplicity = 1 upperMultiplicity = 65535

+reference

CANIF\_SEV\_RX\_ERROR\_DETECTED:

EcucReferenceDef

+destination

lowerMultiplicity = 0 upperMultiplicity = 1

requiresSymbolicNameValue = true

+reference

CANIF\_SEV\_ERRORSTATE\_PASSIVE:

EcucReferenceDef

+destination

lowerMultiplicity = 0 upperMultiplicity = 1

requiresSymbolicNameValue = true

+reference

CANIF\_SEV\_ERRORSTATE\_BUSOFF:

EcucReferenceDef

+destination

lowerMultiplicity = 0 upperMultiplicity = 1

requiresSymbolicNameValue = true

**Figure 10.18: AR\_EcucDef\_CanIfSecurityEventRefs**

*(from IdsM)*

## A - Not applicable requirements

**[SWS\_CANIF\_00999]** *[*These requirements are not applicable to this specifica-tion.*(*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [*SRS\_BSW\_00159*](#_bookmark71) | [*SRS\_BSW\_00167*](#_bookmark73) | [*SRS\_BSW\_00170*](#_bookmark75) | [*SRS\_BSW\_00416*](#_bookmark99) | [*SRS\_BSW\_00168*](#_bookmark74) |
| [*SRS\_BSW\_00423*](#_bookmark101) | [*SRS\_BSW\_00424*](#_bookmark102) | [*SRS\_BSW\_00425*](#_bookmark103) | [*SRS\_-BSW\_00426*](#_bookmark104) | [*SRS\_BSW\_00427*](#_bookmark105) |
| [*SRS\_BSW\_00428*](#_bookmark106) | [*SRS\_BSW\_00429*](#_bookmark107) | [*SRS\_BSW\_00432*](#_bookmark108) | [*SRS\_BSW\_00433*](#_bookmark109) | [*SRS\_BSW\_00336*](#_bookmark87) |
| [*SRS\_BSW\_00417*](#_bookmark100) | [*SRS\_BSW\_00164*](#_bookmark72) | [*SRS\_BSW\_00007*](#_bookmark68) | [*SRS\_BSW\_00307*](#_bookmark78) | [*SRS\_BSW\_00373*](#_bookmark93) |
| [*SRS\_BSW\_00328*](#_bookmark84) | [*SRS\_BSW\_00378*](#_bookmark94) | [*SRS\_BSW\_00306*](#_bookmark77) | [*SRS\_BSW\_00308*](#_bookmark79) | [*SRS\_BSW\_00309*](#_bookmark80) |
| [*SRS\_BSW\_00330*](#_bookmark85) | [*SRS\_BSW\_00172*](#_bookmark76) | [*SRS\_BSW\_00010*](#_bookmark69) | [*SRS\_BSW\_00341*](#_bookmark88) | [*SRS\_BSW\_00334*](#_bookmark86) |
| [*SRS\_Can\_01139*](#_bookmark131) | [*SRS\_Can\_01014*](#_bookmark116) |  |  |  |

)