|  |  |
| --- | --- |
| **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)

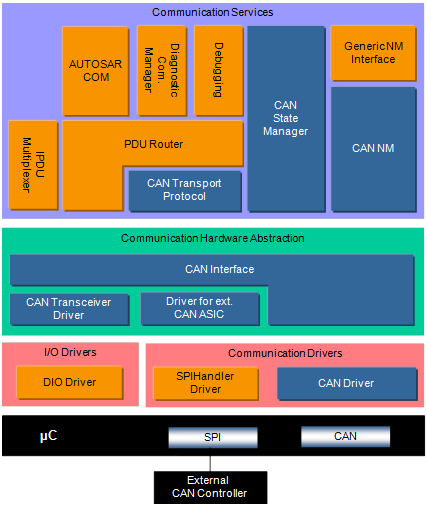
### Introduction and functional overview

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.

|  |
| --- |
| **TÓM TẮT**  1. **CAN Driver**: Cung cấp các API cơ bản để giao tiếp trực tiếp với phần cứng bộ điều khiển CAN như:   * API đọc/ghi thanh ghi bộ điều khiển CAN * API cấu hình bộ điều khiển CAN (tốc độ, chế độ hoạt động, ...) * API gửi/nhận khung dữ liệu CAN   2. **CAN Transceiver Driver**: Cung cấp các API để giao tiếp với phần cứng bộ chuyển đổi CAN như:   * API kích hoạt/vô hiệu hóa bộ chuyển đổi * API điều khiển chế độ ngủ/thức của bộ chuyển đổi   **2. CAN Interface (CanIf)**   * CanIf sử dụng các API của CAN Driver và CAN Transceiver Driver để giao tiếp với phần cứng CAN. * CanIf cung cấp API cao cấp hơn cho các lớp trên như gửi yêu cầu truyền dữ liệu, nhận dữ liệu, xử lý thông báo lỗi. * Các lớp trên như CAN Transport Protocol, PDU Router trong CAN COM Stack gọi các API của CanIf để gửi/nhận dữ liệu qua mạng CAN.   **3. CAN State Manager (CanSM)**   * CanSM quản lý trạng thái của các kênh CAN bằng cách gọi các API của CanIf để khởi động, dừng, đánh thức bộ điều khiển CAN. * CanSM giao tiếp với CAN Network Management (CanNM) để quản lý và đồng bộ hóa trạng thái mạng CAN. * CanSM cũng giao tiếp với ECU State Manager (EcuM) để điều phối khởi động và dừng mạng CAN với trạng thái của ECU.   **4. CAN Network Management (CanNM)**   * CanNM sử dụng các dịch vụ truyền/nhận của CanIf để trao đổi thông tin quản lý mạng với các nút khác trên mạng CAN. * CanNM giao tiếp với CanSM để báo cáo trạng thái mạng CAN và yêu cầu thay đổi trạng thái khi cần thiết. * CanNM cũng giao tiếp với Communication Manager (ComM) để quản lý tình trạng kết nối mạng CAN.   **5. Diagnostic Communication Manager (DCM)**   * DCM sử dụng các dịch vụ của CanIf và CanTrcv để gửi và nhận dữ liệu chẩn đoán qua mạng CAN. * DCM giao tiếp với các mô-đun khác trong hệ thống để thực hiện các chức năng chẩn đoán liên quan đến mạng CAN.   **6. Watchdog Manager (WdgM)**   * WdgM giám sát hoạt động của CAN Driver và CanIf thông qua các watchdog timer. * Nếu phát hiện lỗi, WdgM có thể khởi tạo lại hoặc khởi động lại các thành phần CAN bị lỗi.   **7. End-to-End Communication Protection (E2E)**   * E2E sử dụng các dịch vụ của CanIf để gửi và nhận dữ liệu qua mạng CAN. * E2E cung cấp cơ chế bảo vệ dữ liệu như kiểm tra tính toàn vẹn, phát hiện mất dữ liệu, đảm bảo dữ liệu được truyền đầy đủ từ đầu đến đầu.   **8. ECU State Manager (EcuM)**   * Quản lý trạng thái của các ECU (Electronic Control Unit) trong hệ thống. * EcuM phối hợp với CanSM để đảm bảo ECU được khởi động và dừng đúng cách, bao gồm cả việc khởi tạo và dừng các mạng giao tiếp như CAN.   9. **Communication Manager (ComM)**   * Quản lý giao thức giao tiếp trên các mạng khác nhau, bao gồm cả CAN. * ComM phối hợp với CanNM để quản lý tình trạng kết nối mạng CAN và điều khiển việc tham gia hoặc rời khỏi mạng. |



**Figure 1.1:** **AUTOSAR CAN Layer Model (see [**[**7**](#_bookmark42)**])**

Mô-đun CAN Interface bao gồm tất cả các tác vụ độc lập với phần cứng CAN, thuộc về các trình điều khiển thiết bị giao tiếp CAN của ECU tương ứng. Chức năng đó được triển khai trong mô-đun CAN Interface, để các trình điều khiển thiết bị CAN bên dưới chỉ tập trung vào việc truy cập và điều khiển thiết bị phần cứng CAN cụ thể.

CanIf đáp ứng các yêu cầu chính về luồng điều khiển và luồng dữ liệu của PDU Router và các mô-đun giao tiếp lớp trên của AUTOSAR COM stack: transmit request processing, transmit confirmation / receive indication / error notification and start / stop of a CAN Controller and thus waking up / participating on a network. API xử lý và thông báo dữ liệu của nó dựa trên CAN L-SDU, trong khi các API dành cho điều khiển và xử lý chế độ cung cấp một quan điểm liên quan đến Bộ điều khiển CAN.

Trong trường hợp có ***Yêu cầu Truyền***, CanIf hoàn tất việc truyền L-PDU với các tham số tương ứng và chuyển tiếp L-PDU CAN qua CanDriver thích hợp đến Bộ điều khiển CAN. Khi nhận, CanIf phân phối các L-PDU Received dưới dạng L-SDU cho lớp trên. Việc phân công giữa L-SDU Received và lớp trên được cấu hình tĩnh. Tại xác nhận truyền, CanIf chịu trách nhiệm thông báo cho các lớp trên về việc truyền thành công.

Mô-đun CAN Interface cung cấp quyền truy cập trừu tượng giao tiếp CAN cho các dịch vụ của CAN Driver và CAN Transceiver Driver để điều khiển và giám sát mạng CAN. CAN Interface chuyển tiếp xuống các yêu cầu thay đổi trạng thái từ CAN State Manager đến các trình điều khiển thiết bị CAN ở lớp thấp hơn, và chuyển tiếp lên các sự kiện từ CAN Driver / CAN Transceiver Driver bởi mô-đun CAN Interface đến ví dụ mô-đun NM tương ứng.

### Acronyms and Abbreviations

Danh mục thuật ngữ dưới đây bao gồm các từ viết tắt và viết tắt liên quan đến mô-đun CAN Interface mà không được đưa vào trong [8, Bảng chú giải AUTOSAR].

|  |  |
| --- | --- |
| **Abbreviation** | **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 hardware device. One CAN Controller is connected to one physical channel. |
| CAN Device Driver | Generic term of CAN Driver and CAN Transceiver Driver. |
| CAN Hardware Unit | A CAN Hardware Unit may consist of one or multiple CAN Controllers of the same type and one, two or multiple CAN RAM areas. The CAN Hardware Unit is located on-chip or as 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 presence 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 network. Usually they are handled by one upper layer module. |
| Outer priority inversion | A time gap occurs between two consecutive transmit L-PDUs. In this case a lower priority L-PDU from another node can prevent sending the own higher priority L-PDU. Here the higher priority L- PDU cannot participate in arbitration during network access 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 cung cấp một Đặc tả Chung về các mô-đun Phần mềm Cơ bản [9, SWS BSW General], cũng áp dụng cho Giao diện CAN.

Do đó, đặc tả SWS BSW General nên được xem xét là một đặc tả bổ sung và bắt buộc cho Giao diện CAN.

### Constraints and assumptions

#### Limitations

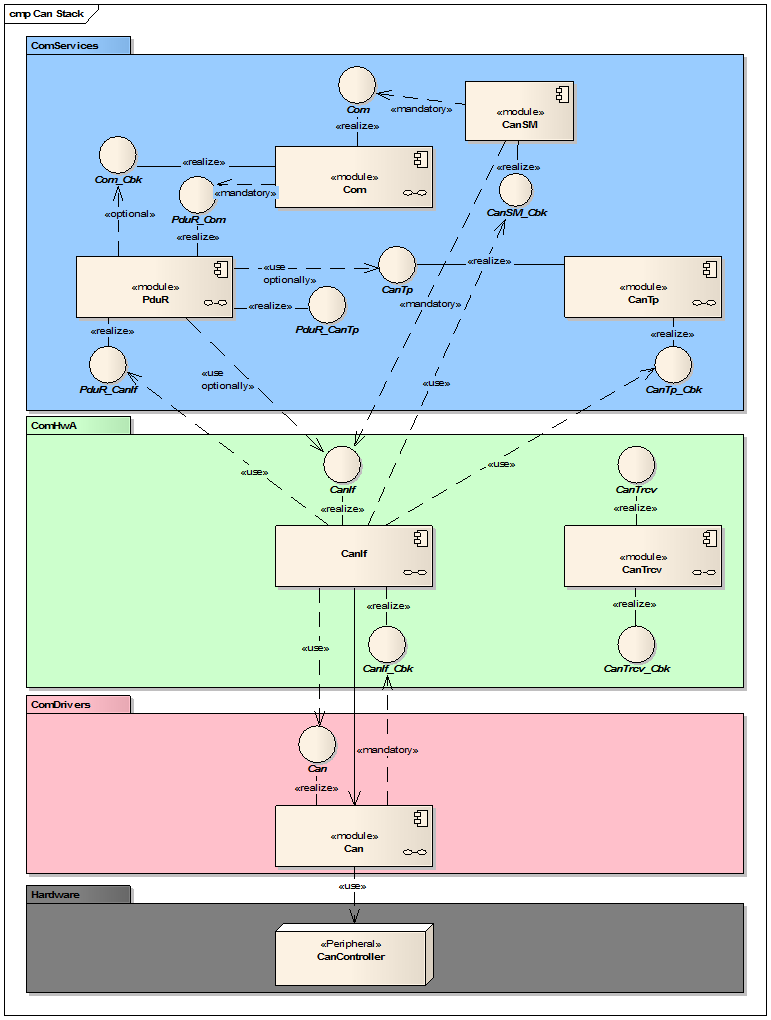
CAN Interface chỉ có thể được sử dụng cho giao tiếp CAN và được thiết kế đặc biệt để hoạt động với một hoặc nhiều CAN Driver và CAN Transceiver Driver bên dưới. Nhiều mô-đun CAN Driver khác nhau tương ứng với các Đơn vị Phần cứng CAN khác nhau được đại diện bởi một giao diện chung duy nhất như được quy định trong CAN Driver specification [[1](#_bookmark36)]. Tương tự, nhiều mô-đun CAN Transceiver Driver khác nhau tương ứng với các thiết bị CAN Transceiver khác nhau cũng được đại diện bởi một giao diện chung duy nhất như được quy định trong CAN Transceiver Driver specification [[2](#_bookmark37), Specification of CAN Transceiver Driver]. Các giao thức khác ngoài CAN (ví dụ: LIN hoặc FlexRay) không được hỗ trợ.

Lưu ý rằng một bộ lọc PnTxFilter đang hoạt động đảm bảo rằng thông điệp đầu tiên trên đường truyền là CanIfTxPduPnFilterPdu. Trong trường hợp CanIfTxPduPnFilterPdu là PDU NM, bộ xếp tầng COM-Stack sẽ đảm bảo rằng các Nhóm PDU bị vô hiệu hóa cho đến khi PDU đó được gửi đi thành công. Tuy nhiên, các yêu cầu gửi PDU khác (ví dụ: PDU khởi động ban đầu, TP-PDU, XCP-PDU) sẽ bị từ chối cho đến khi PDU được cấu hình đã được gửi đi. Chỉ PDU đầu tiên khởi tạo quá trình Thức Mạng mới là CanIfTxPduPnFilterPdu. Trong trường hợp giao tiếp đang diễn ra và có sự nhận thành công khung tin với PnTxFilter được bật, PnTxFilter sẽ bị vô hiệu hóa. Trong trường hợp này, PnTxFilter không cần thiết vì sẽ nhận được phản hồi từ một Nút đã hoạt động.

#### Applicability to car domains

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

### Dependencies to other modules

Phần này mô tả các mối quan hệ với các mô-đun khác trong kiến trúc phần mềm cơ bản AUTOSAR. Nó chứa các mô tả ngắn gọn về thông tin cấu hình và các dịch vụ mà Lớp CAN Interface yêu cầu từ các mô-đun khác (see [Figure](#_bookmark55) [5.1](#_bookmark55)).

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

|  |
| --- |
| Hình 5.1 này minh họa các mối phụ thuộc và mối quan hệ của mô-đun CAN Interface (CanIf) với các mô-đun khác trong kiến trúc phần mềm cơ bản AUTOSAR (AUTOSAR BSW).  1. CAN Communication Stack (CAN COM Stack):  - Phần này bao gồm các mô-đun ComServices, Com, CanSM, CanTp, PduR và PduR\_CanTp.  - Các mô-đun này sử dụng (realize) các dịch vụ của CanIf để gửi/nhận dữ liệu qua giao thức CAN.  - CanSM (CAN State Manager) là mô-đun bắt buộc sử dụng dịch vụ của CanIf để quản lý trạng thái của mạng CAN.  2. CanTxA:  - Đây là một lớp trừu tượng cung cấp giao diện chung để các mô-đun khác sử dụng dịch vụ của CanIf.  3. CanIf:  - Mô-đun CAN Interface cung cấp giao diện để giao tiếp với các trình điều khiển thiết bị CAN như Can và CanTrcv.  - CanIf sử dụng (use) dịch vụ của Can và CanTrcv.  4. Can:  - Đây là mô-đun bắt buộc, đại diện cho CAN Driver, giao tiếp trực tiếp với phần cứng Bộ điều khiển CAN.  5. CanTrcv:  - Mô-đun này đại diện cho CAN Transceiver Driver, giao tiếp với phần cứng Bộ chuyển đổi CAN.  6. Can Driver:  - Đây là lớp thấp nhất, gồm các trình điều khiển phần cứng cấp thấp giao tiếp trực tiếp với phần cứng Bộ điều khiển CAN.  Tóm lại, các mô-đun trong CAN COM Stack sử dụng dịch vụ của CanIf để giao tiếp qua mạng CAN. CanIf đóng vai trò trung gian, sử dụng dịch vụ của Can và CanTrcv để giao tiếp với phần cứng Bộ điều khiển CAN và Bộ chuyển đổi CAN. Các mối quan hệ này thể hiện cách các thành phần phần mềm và phần cứng CAN liên kết với nhau trong kiến trúc AUTOSAR. |

#### Upper Protocol Layers

Trong kiến trúc phần mềm cơ bản AUTOSAR, các lớp trên của mô-đun CAN Interface (Viết tắt: CanIf) được đại diện bởi mô-đun PDU Router (Viết tắt: PduR), mô-đun CAN Network Management (Viết tắt: CanNm), mô-đun CAN Transport Layer (Viết tắt: CanTp), mô-đun CAN State Manager (Viết tắt: CanSm), mô-đun ECU State Manager (Viết tắt: EcuM), mô-đun Complex Driver (Viết tắt: \_CDD\_), mô-đun Universal Calibration Protocol (Viết tắt: \_XCP\_), Global Time Synchronization over CAN (Viết tắt: CanTSyn), mô-đun J1939 Transport Layer (Viết tắt: J1939Tp) và mô-đun J1939 Network Management (Viết tắt: J1939Nm).

Kiến trúc phần mềm cơ bản AUTOSAR cho thấy các bộ đệm dữ liệu của ứng dụng được đặt trong lớp trên mà chúng thuộc về. Việc truy cập trực tiếp vào các bộ đệm này bị cấm. Vị trí của bộ đệm được CanIf chuyển từ hoặc đến mô-đun CAN Driver (Viết tắt: CanDrv) trong quá trình truyền và nhận. Trong quá trình thực thi các dịch vụ chỉ báo truyền/nhận này, vị trí của bộ đệm được chuyển. Tính toàn vẹn dữ liệu được đảm bảo bằng cách sử dụng cơ chế khóa mỗi lần truy cập bộ đệm. Xem phần 7.17 "Tính toàn vẹn dữ liệu".

API được CanIf sử dụng bao gồm các dịch vụ thông báo như các tác nhân cơ bản để chuyển dữ liệu liên quan đến CAN (ví dụ: Độ dài Dữ liệu) đến lớp trên đích. Các tham số gọi của các dịch vụ này trỏ đến thông tin được đệm trong CanDrv hoặc chúng tham chiếu trực tiếp đến Phần cứng CAN.

Ngoài ra, CanIf hỗ trợ một callout đến mô-đun Bus Mirroring, để báo cáo nội dung của các khung dữ liệu đã nhận và đã gửi.

#### Initialization: Ecu State Manager

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

#### Mode Control: CAN State Manager

Module CanSm chịu trách nhiệm quản lý điều khiển chế độ của tất cả các bộ điều khiển CAN và bộ truyền CAN được hỗ trợ.

#### Lower layers: CAN Driver

Trình điều khiển thiết bị lớp dưới chính của CAN được đại diện bởi CanDrv (see [[1](#_bookmark36), Specifi- cation of CAN Driver]). CanIf có mối quan hệ chặt chẽ với CanDrv do vị trí của nó trong AUTOSAR Basic Software Architecture.

CanDrv cung cấp truy cập trừu tượng phần cứng chỉ đến Bộ điều khiển CAN, nhưng việc kiểm soát các chế độ hoạt động chỉ được thực hiện trong CanSm. CanDrv phát hiện và xử lý các sự kiện của các Bộ điều khiển CAN và thông báo những sự kiện này đến CanIf. CanIf chuyển các yêu cầu chế độ hoạt động của CanSm đến các Bộ điều khiển CAN tương ứng bên dưới.

CanDrv cung cấp một L-PDU chuẩn hóa để đảm bảo CanIf không phụ thuộc vào phần cứng. Con trỏ đến L-PDU chuẩn hóa này trỏ hoặc đến bộ đệm tạm thời (ví dụ: chuẩn hóa dữ liệu) hoặc đến CanDrv phụ thuộc vào phần cứng CAN. Đối với CanIf, loại bộ đệm L-PDU là vô hình.

CanIf cung cấp các dịch vụ thông báo được sử dụng bởi CanDrv trong tất cả các kịch bản thông báo, ví dụ: xác nhận truyền (mục 8.4.2 “CanIf\_TxConfirmation”, xem [SWS\_CANIF\_00007]), chỉ báo nhận (mục 8.4.3 “CanIf\_RxIndication”, xem [SWS\_CANIF\_00006]) và thông báo thay đổi chế độ bộ điều khiển (mục 8.4.8, xem [SWS\_CANIF\_00699]).

Trong trường hợp sử dụng nhiều CanDrv phục vụ các vectơ ngắt khác nhau, các dịch vụ gọi lại nêu trên phải có khả năng tái nhập, tham khảo mục 7.24 “Hỗ trợ Trình điều khiển CAN Nhiều”. Khả năng tái nhập của các hàm gọi lại được quy định trong mục 8.4.

Các dịch vụ gọi lại do CanDrv gọi được khai báo và triển khai bên trong CanIf. Các dịch vụ gọi lại do CanIf gọi được khai báo và đặt bên trong lớp dịch vụ truyền thông trên, ví dụ như PduR, CanNm, CanTp. Cấu trúc CanIf được quy định trong mục 5.7 “Cấu trúc tệp”.

Số lượng các Bộ điều khiển CAN được cấu hình không nhất thiết tương ứng với số lượng Bộ truyền CAN được sử dụng. Trong trường hợp nhiều Bộ điều khiển CAN của các loại khác nhau hoạt động trên cùng một mạng CAN, một Bộ truyền CAN và CanTrcv là đủ, trong khi đó phụ thuộc vào loại thiết bị Bộ điều khiển CAN, cần một hoặc hai CanDrv khác nhau (see [section](#_bookmark149) [7.5](#_bookmark149) “[Physical channel view](#_bookmark149)”).

#### Lower layers: CAN Transceiver Driver

Trình điều khiển thiết bị tầng thấp thứ hai có sẵn của CAN được biểu diễn bởi CanTrcv (xem [2, Specification of CAN Transceiver Driver]).

Mỗi CanTrcv tự nó điều khiển các chế độ hoạt động của thiết bị CAN Transceiver. CanIf chỉ ánh xạ tất cả các API của nhiều CanTrcv bên dưới thành một API duy nhất, do đó CanSm có thể kích hoạt một chuyển đổi của các chế độ CAN Transceiver tương ứng. Không có chức năng điều khiển hay xử lý nào thuộc về CanTrcv được thực hiện bên trong CanIf.

CanIf ánh xạ các dịch vụ sau của tất cả các CanTrcv bên dưới thành một giao diện duy nhất. Các dịch vụ này được mô tả chi tiết hơn trong CAN Transceiver Driver SWS (xem [2, Specification of CAN Transceiver Driver]):

- Dịch vụ yêu cầu và đọc chế độ CanTrcv duy nhất để quản lý các chế độ hoạt động của mỗi thiết bị CAN Transceiver bên dưới.

- Dịch vụ đọc lý do thức dậy của CAN Transceiver.

- Dịch vụ yêu cầu chế độ để kích hoạt/vô hiệu hóa/xóa trạng thái sự kiện thức dậy của mỗi CAN Transceiver đang sử dụng (CanIf\_SetTrcvMode(), xem [SWS\_CANIF\_00287]).

#### Configuration

Thiết kế CanIf được tối ưu hóa để quản lý các khả năng đặc thù của giao thức CAN và xử lý Bộ điều khiển CAN bên dưới được sử dụng. CanIf có khả năng thay đổi cấu hình CAN mà không cần phải xây dựng lại. Do đó, hàm CanIf\_Init() (xem [SWS\_CANIF\_00001]) lấy thông tin cấu hình CAN cần thiết từ các bộ chứa và tham số cấu hình, được chỉ định (liên kết làm tham chiếu hoặc các tham số bổ sung) trong chương 10, xem Hình 10.1.

Phần này cung cấp tóm tắt về thông tin đã lấy, ví dụ:

- Số lượng Bộ điều khiển CAN. Số lượng Bộ điều khiển CAN là cần thiết để phân phát các L-PDU truyền và nhận và để kiểm soát trạng thái của các Trình điều khiển CAN hiện có (xem CanIfCtrlDrvCfg).

- Số lượng Tay cầm Đối tượng Phần cứng (HTH). Để giám sát các yêu cầu truyền, Giao diện CAN cần biết số lượng HTH và sự gán giữa mỗi HTH và Bộ điều khiển CAN tương ứng (xem CanIfHthCanCtrlIdRef; CanIfHthIdSymRef).

- Phạm vi các ID CAN nhận được đi qua bộ lọc chấp nhận phần cứng cho mỗi đối tượng phần cứng. Giao diện CAN sử dụng các sự gán cố định giữa HRH và L-PDU được nhận trong đối tượng phần cứng tương ứng để thực hiện thuật toán tìm kiếm (xem mục 7.20 “Bộ lọc nhận phần mềm”, xem CanIfHrhSoftwareFilter, CanIfHrhCanCtrlIdRef, CanIfHrhIdSymRef).

CanIf cần thông tin về tất cả các lớp dịch vụ truyền thông phía trên được sử dụng và L-SDU để phân phát. Thông tin sau phải được thiết lập tại thời điểm cấu hình để tích hợp CanIf vào ngăn xếp COM của AUTOSAR:

- Mô-đun lớp trên truyền và I-PDU truyền cho mỗi L-SDU truyền.

=> Được sử dụng để phân phát các dịch vụ xác nhận truyền (xem CanIfTxPduId).

- Mô-đun lớp trên nhận và I-PDU nhận cho mỗi L-SDU nhận.

=> Được sử dụng để phân phát L-SDU trong quá trình chỉ báo nhận (xem CanIfRxPduId).

CanIf cần mô tả về bộ điều khiển và ECU của chính nó, cái được kết nối với một hoặc nhiều mạng CAN. Thông tin sau được lấy từ ma trận truyền thông CAN, một phần của cấu hình hệ thống AUTOSAR (xem CanIfTxPduCfg, CanIfRxPduCfg):

- Tất cả L-PDU nhận được trên mỗi kênh vật lý của ECU này.

=> Được sử dụng để lọc phần mềm và phân phát L-SDU nhận.

- Tất cả các L-SDU sẽ được truyền bởi mỗi kênh vật lý trên ECU này.

=> Được sử dụng cho yêu cầu truyền và phân phát L-PDU truyền.

- Thuộc tính của các L-PDU này (ID, Độ dài Dữ liệu).

=> Được sử dụng cho lọc phần mềm, dịch vụ chỉ báo nhận, kiểm tra Độ dài Dữ liệu.

- Bộ truyền cho mỗi L-SDU truyền (tức là PduR, CanNm, CanTp).

=> Được sử dụng cho các dịch vụ xác nhận truyền.

- Bộ nhận cho mỗi L-SDU nhận (tức là PduR, CanNm, CanTp).

=> Được sử dụng cho phân phát L-PDU.

- Tên tượng trưng của L-PDU/L-SDU.

=> Được sử dụng để biểu diễn địa chỉ bộ đệm dữ liệu Rx/Tx.

#### File structure

##### Code file structure

[SWS\_CANIF\_00378] [CanIf sẽ truy cập vị trí của API của tất cả các CanDrv bên dưới được sử dụng cho cấu hình thời gian liên kết bằng một tập hợp các con trỏ hàm cho mỗi CanDrv.] ()

Các giá trị cho các con trỏ hàm cho mỗi CanDrv được cung cấp tại thời điểm liên kết.

##### Header file structure

[SWS\_CANIF\_00672] [Tệp tiêu đề CanIf.h chỉ chứa các khai báo extern của các hằng số, dữ liệu toàn cục và các dịch vụ được chỉ định trong CanIf.] ()

Các hằng số, kiểu dữ liệu toàn cục và các hàm chỉ được sử dụng nội bộ bởi CanIf, được khai báo trong CanIf.c.

[SWS\_CANIF\_00903] [CanIf phải bao gồm tệp tiêu đề Mirror.h nếu hỗ trợ Phản chiếu Bus được kích hoạt (xem CanIfBusMirroringSupport).](SRS\_Can\_01172)

### Requirements Tracing

Các bảng sau đây tham chiếu các yêu cầu được chỉ định trong [10] cũng như [11] và liên kết đến việc thực hiện những yêu cầu này. Xin lưu ý rằng nếu cột 'Được đáp ứng bởi' trống đối với một yêu cầu cụ thể, điều này có nghĩa là yêu cầu đó không được tài liệu này đáp ứng.

|  |  |  |
| --- | --- | --- |
| **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

Các dịch vụ của CanIf có thể được chia thành các nhóm chính sau:

* Khởi tạo
* Dịch vụ yêu cầu truyền
* Dịch vụ xác nhận truyền
* Dịch vụ chỉ báo nhận
* Dịch vụ kiểm soát chế độ bộ điều khiển
* Dịch vụ kiểm soát chế độ PDU

**Các ứng dụng có thể của CanIf:**

***i. Chế độ Ngắt***

CanDrv xử lý các ngắt do Bộ điều khiển CAN kích hoạt. CanIf, dựa trên sự kiện, được thông báo khi có sự kiện xảy ra. Trong trường hợp này, các dịch vụ liên quan của CanIf được gọi trong các ISR tương ứng trong CanDrv.

***ii. Chế độ Thăm dò***

CanDrv được kích hoạt bởi SchM và thực hiện các quá trình tiếp theo (Chế độ Thăm dò). Trong trường hợp này, Can\_MainFunction\_<Write/Read/BusOff/Wakeup/Transceiver>() phải được gọi định kỳ trong khoảng thời gian xác định.

CanIf được thông báo bởi CanDrv về các sự kiện (Nhận, Truyền, BusOff, Hết thời gian) đã xảy ra trong một trong các Bộ điều khiển CAN, tương tự như hoạt động theo ngắt. CanDrv chịu trách nhiệm cập nhật thông tin tương ứng liên quan đến sự kiện đã xảy ra trong Bộ điều khiển CAN, ví dụ như nhận một L-PDU.

***iii. Chế độ Hỗn hợp: CanDrv dựa trên ngắt và thăm dò***

Chức năng có thể được chia giữa chế độ hoạt động dựa trên ngắt và chế độ hoạt động dựa trên thăm dò tùy thuộc vào các Bộ điều khiển CAN được sử dụng. Ví dụ: Nhận FullCAN dựa trên thăm dò và nhận BasicCAN dựa trên ngắt, truyền dựa trên thăm dò và nhận dựa trên ngắt, v.v.

Đặc tả này mô tả một giao diện duy nhất, có giá trị cho cả ba loại chế độ hoạt động. Tóm lại, CanIf hoạt động theo cùng một cách, cho dù bất kỳ sự kiện nào được xử lý theo ngắt, mức tác vụ hoặc hỗn hợp. Sự khác biệt duy nhất là ngữ cảnh cuộc gọi và có thể là cách ngắt thông báo: tiền xử lý hoặc hợp tác. Tất cả các dịch vụ được thực hiện theo cấu hình.

Các đoạn sau đây mô tả chức năng của CanIf.

#### Hardware object handles

Hardware Object Handles – (HOH) cho truyền (HTH) cũng như cho nhận (HRH) đại diện cho một tham chiếu trừu tượng đến CAN mailbox structure, chứa các tham số liên quan đến CAN như CanId, DLC và dữ liệu. Dựa trên sự trừu tượng hóa bộ đệm phần cứng CAN, mỗi Đối tượng Phần cứng được tham chiếu trong CanIf độc lập với bố trí bộ đệm phần cứng CAN. HOH được sử dụng làm tham số trong các cuộc gọi dịch vụ giao diện của CanDrv và được cung cấp bởi cấu hình của CanDrv và được CanDrv sử dụng làm định danh cho các bộ đệm giao tiếp của hộp thư CAN.

CanIf chỉ đóng vai trò là người dùng của Hardware Object Handles, nhưng không diễn giải nó dựa trên thông tin cụ thể của phần cứng. Do đó, CanIf vẫn độc lập với phần cứng.

[**SWS\_CANIF\_00023**] [CanIf phải tránh truy cập trực tiếp vào các bộ đệm giao tiếp cụ thể của phần cứng và phải truy cập chúng thông qua các dịch vụ giao diện của CanDrv.](SRS\_Can\_01001)

Lý do cho [**SWS\_CANIF\_00023**]: CanIf vẫn độc lập với phần cứng vì các giao diện của CanDrv được gọi bằng các tham số HOH, giúp trừu tượng hóa các thuộc tính bộ đệm phần cứng CAN cụ thể.

Mỗi Bộ điều khiển CAN có thể cung cấp nhiều CAN Transmit Hardware Objects trong CAN Mailbox. Chúng có thể được liên kết logic với một toàn bộ nhóm Đối tượng Phần cứng (Đối tượng Phần cứng đa kênh) và do đó được định địa chỉ bởi một HTH.

[**SWS\_CANIF\_00662**] [CanIf phải sử dụng hai loại HOH để cho phép truy cập vào CanDrv:

* Hardware Transmit Handle (HTH)
* Hardware Receive Handle (HRH).

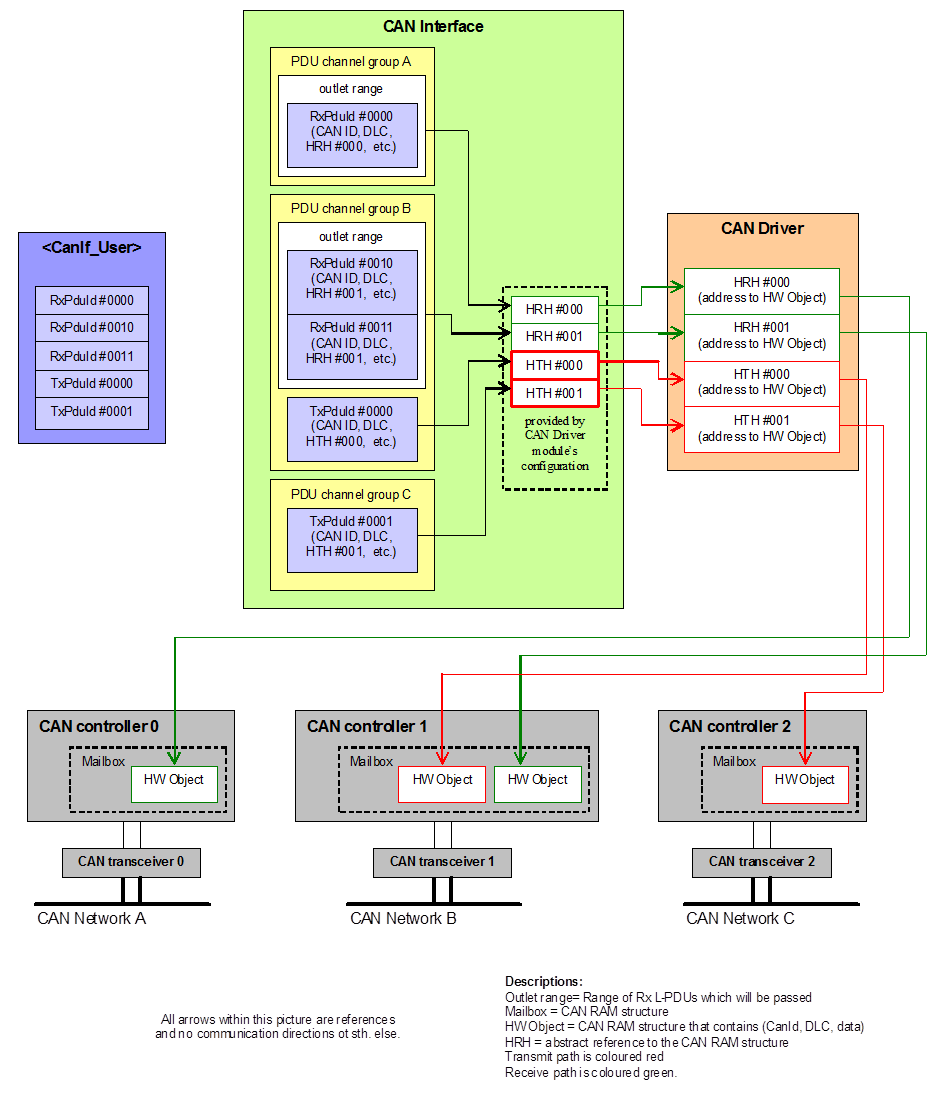
[**SWS\_CANIF\_00291**] [Định nghĩa của HRH: HRH phải là Hardware Receive Object của hộp thư Bộ điều khiển CAN.]()

[SWS\_CANIF\_00665] [HRH phải cho phép CanIf sử dụng phương pháp nhận BasicCAN hoặc FullCAN của đơn vị nhận tham chiếu và chỉ báo một L-SDU Đã Nhận đến mô-đun lớp trên đích.]()

[SWS\_CANIF\_00663] [Nếu HRH tham chiếu một đơn vị nhận được cấu hình cho nhận BasicCAN, lọc phần mềm phải được kích hoạt trong CanIf.]()

[SWS\_CANIF\_00664] [Nếu sử dụng nhiều HRH, mỗi HRH phải thuộc ít nhất một nhóm đơn hoặc cố định của Rx L-SDU (CanRxPduIds).]()

HRH có thể được cấu hình để nhận:

* Một CanId đơn (FullCAN)
* Một nhóm các CanId đơn (BasicCAN)
* Một phạm vi/khu vực của CanId (BasicCAN) hoặc
* Tất cả các CanId.

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

[**SWS\_CANIF\_00292**] [Định nghĩa của HTH: HTH phải là Hardware Transmit Object của hộp thư Bộ điều khiển CAN.]()

[**SWS\_CANIF\_00666**] [HTH phải cho phép CanIf sử dụng phương pháp truyền BasicCAN hoặc FullCAN của đơn vị truyền tham chiếu và xác nhận một L-SDU đã truyền đến một mô-đun lớp trên đích.]()

**[SWS\_CANIF\_00466**] [Mỗi CanIf Tx L-PDU phải được gán tĩnh cho một bộ chứa cấu hình CanIfBufferCfg vào thời điểm cấu hình (xem CanIfTxPduBufferRef).]()

Lý do cho [**SWS\_CANIF\_00466**]: Các CanIf Tx L-PDU không tham chiếu HTHs, nhưng CanIfBufferCfg, mà ngược lại tham chiếu HTHs.

[**SWS\_CANIF\_00667**] [Nếu sử dụng nhiều HTHs, mỗi HTH phải thuộc ít nhất một nhóm đơn hoặc cố định của Tx L-PDU (CanTxPduIds).]()

[**SWS\_CANIF\_00115**] [CanIf phải có khả năng sử dụng tất cả HRHs và HTHs của một CanDrv như một khu vực đánh số chung, duy nhất bắt đầu từ không.]()

Các HRHs và HTHs dành riêng được suy ra từ bộ cấu hình của CanDrv. Định nghĩa của HTH/HRH trong khu vực đánh số và Đối tượng Phần cứng là trách nhiệm của CanDrv.

#### Static L-PDUs

CanIf cung cấp truy cập chung đến dữ liệu liên quan đến L-SDU CAN cho các lớp trên. Các thuộc tính của bảng sau được biểu diễn dưới dạng các tham số cấu hình và được chỉ định trong chương 10:

|  |  |
| --- | --- |
| **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 hỗ trợ kích hoạt và vô hiệu hóa tất cả các L-PDU thuộc về một Bộ điều khiển CAN cho truyền cũng như cho nhận (xem 7.19.2, xem CanIf\_SetPduMode(), [SWS\_CANIF\_00008]). Đối với việc kiểm soát chế độ L-PDU, xin tham khảo mục 7.19.

Mỗi L-PDU được liên kết với một mô-đun lớp trên để đảm bảo việc phân phối chính xác trong quá trình nhận, xác nhận truyền và truy cập dữ liệu. Mỗi mô-đun lớp trên có thể sử dụng các L-PDU để phục vụ các Bộ điều khiển CAN khác nhau đồng thời.

Theo kiến trúc PDU được định nghĩa cho toàn bộ ngăn xếp giao tiếp AUTOSAR (xem [7, Kiến trúc Phần mềm Lớp]), việc sử dụng L-PDU được chia thành hai cách khác nhau:

* Đối với yêu cầu truyền và API thăm dò truyền/nhận, mô-đun lớp trên sử dụng ID L-SDU (CanTxPduId/CanRxPduId) được định nghĩa bởi CanIf như là tham số.
* Đối với tất cả các API gọi lại, mà được gọi bởi CanIf tại các mô-đun lớp trên, CanIf truyền PduId mục tiêu được định nghĩa bởi mỗi mô-đun lớp trên làm tham số.

Nguyên tắc là người gọi phải sử dụng Id L-PDU/L-SDU mục tiêu được định nghĩa của người được gọi.

Nếu quá trình khởi động không được thực hiện và lớp trên thực hiện yêu cầu truyền tới CanIf, không có L-SDUs được truyền xuống lớp dưới và DET sẽ được kích hoạt. Do đó, không có dữ liệu chưa được khởi tạo nào có thể được truyền trên mạng. Hành vi của hàm truyền L-PDU/L-SDU được chỉ định chi tiết trong mục 8.3.6.

#### Dynamic L-PDUs

CanIf phải hỗ trợ khả năng lọc các tin nhắn đến bằng cách sử dụng CanIfRxPduCanIdMask. Việc lọc sẽ được thực hiện bằng cách so sánh CanId đến với CanIfRxPduCanId được lưu trữ sau khi áp dụng CanIfRxPduCanIdMask cho cả hai ID. Điều này nên được thực hiện sau khi lọc các CanId thông thường mà không có mặt nạ, để cho phép xử lý riêng biệt của một số CanId nằm trong phạm vi được xác định bởi mặt nạ hoặc một phạm vi dựa trên CanId.

Ngoài ra, DYNAMIC Tx và Rx L-SDUs phải được hỗ trợ, trong đó CanId nằm trong MetaData của L-SDU.

Trong quá trình truyền L-SDUs động, khi được xác định một CanIfTxPduCanIdMask, các phần biến của CanId được cung cấp qua MetaData phải được hợp nhất với CanId bằng cách sử dụng mặt nạ này. Khi không có CanIfTxPduCanIdMask và không có CanIfTxPduCanId được cấu hình, MetaData phải được sử dụng trực tiếp làm CanId.

Trong quá trình nhận L-SDUs động, CanId nhận được phải được đặt trong MetaData của L-SDU. Nội dung của MetaData là độc lập với tham số CanIfRxPduCanIdMask.

[**SWS\_CANIF\_00844**] [CanIf phải hỗ trợ các L-PDU động, trong đó CanId hoặc các phần liên quan của CanId được đặt trong MetaData của một L-SDU.]()

##### Dynamic Transmit L-PDUs

Định nghĩa của các Transmit L-PDU động: Các L-PDU cho phép việc cấu hình lại CanId trong quá trình chạy (CanIfTxPduType) hoặc nơi mà ID hoặc các phần của nó được cung cấp dưới dạng MetaData của L-SDU.

Việc sử dụng tất cả các phần tử L-PDU khác đều giống như các Transmit L-PDU tĩnh bình thường:

* Thông báo xác nhận truyền CanIfTxPduUserTxConfirmationUL không thể được cấu hình lại vì nó thuộc về L-PDU.
* Chiều dài dữ liệu và con trỏ đến bộ đệm dữ liệu đều được xác định bởi mô-đun lớp trên khi gọi hàm CanIf\_Transmit().

Hàm CanIf\_SetDynamicTxId() (xem [SWS\_CANIF\_00189]) cấu hình lại CanId của một L-PDU động với CanIfTxPduType.

[**SWS\_CANIF\_00188**] [CanIf phải xử lý hai bit cao nhất của CanId (xem [1, Đặc tả của Driver CAN], định nghĩa của Can\_IdType [SWS\_Can\_00416]) để xác định loại CanId được sử dụng và do đó cách thức truyền L-PDU động sẽ được thực hiện.]()

[**SWS\_CANIF\_00673**] [CanIf phải đảm bảo tính nhất quán dữ liệu của CanId trong trường hợp chạy hàm CanIf\_SetDynamicTxId(). Dịch vụ này có thể bị gián đoạn bởi cuộc gọi của CanIf\_Transmit() ảnh hưởng đến cùng một L-PDU, xem [SWS\_CANIF\_00064].]()

[SWS\_CANIF\_00855] [Nếu CanIfTxPduCanIdMask và CanIfTxPduCanId được bỏ qua, CanId sẽ được lấy trực tiếp từ MetaData.]()

[SWS\_CANIF\_00856] [CanIfTxPduCanIdMask sẽ được bỏ qua khi cấu hình meta dữ liệu không chứa CAN\_ID\_32 cho L-SDU này.]()

[SWS\_CANIF\_00854] [Nếu Mục MetaData CAN\_ID\_32, CanIfTxPduCanIdMask và CanIfTxPduCanId có sẵn, CanIfTxPduCanIdMask xác định các bit trong CanIfTxPduCanId và các bit của Can\_IdType được tạo ra từ CanIfTxPduCanIdType mà sẽ xuất hiện trong CanId thực tế, các bit khác được lấy từ MetaData.]()

Lưu ý: CanId kết quả có thể được tính như sau: (CanIfTxPduCanId & CanIfTxPduCanIdMask) | (<các phần ID động> & ∼CanIfTxPduCanIdMask)

[SWS\_CANIF\_00857] [CanIf\_Init() (xem [SWS\_CANIF\_00085]) khởi tạo các CanId của các Transmit L-PDU động với CanIfTxPduType đến giá trị được cấu hình qua CanIfTxPduCanId.]()

##### Dynamic receive L-PDUs

Định nghĩa của các Receive L-PDU động: Các L-PDU tương ứng với một tập hợp các CanId, trong đó CanId thực tế được nhận được cung cấp cho các lớp trên như một phần của dữ liệu PDU.

[SWS\_CANIF\_00847] [Cấu hình phải đảm bảo rằng các Receive L-PDU động sử dụng một phạm vi ID hoặc một mặt nạ và rằng MetaDataItem CAN\_ID\_32 được cấu hình cho L-SDU. Bên cạnh đó, việc lọc phần mềm phải được kích hoạt cho các L-SDU này.]()

[SWS\_CANIF\_00848] [Khi nhận được một L-SDU động, CanIf phải đặt CanId vào MetaDataItem của loại CAN\_ID\_32.]()

#### Physical channel view

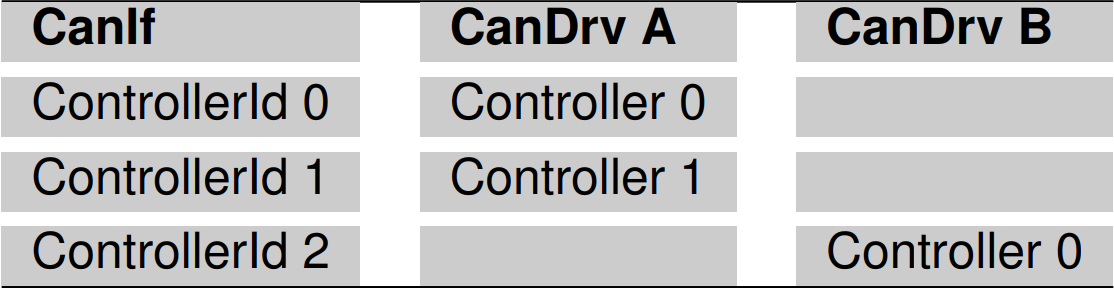
Một kênh vật lý được liên kết với một Bộ điều khiển CAN và một Bộ truyền CAN, trong khi một hoặc nhiều kênh vật lý có thể được kết nối với một mạng duy nhất.

CanIf cung cấp các dịch vụ để điều khiển tất cả các thiết bị CAN như Bộ điều khiển CAN và Bộ truyền CAN của tất cả các ECU được hỗ trợ trên kênh CAN. Các API này được sử dụng bởi CanSm để cung cấp một cái nhìn mạng đến ComM (xem [3]) được sử dụng để thực hiện yêu cầu thức dậy và ngủ cho tất cả các kênh vật lý kết nối với một mạng duy nhất.

CanIf chuyển thông tin trạng thái được cung cấp bởi CanDrv và CanTrcv riêng biệt cho mỗi kênh vật lý như thông tin trạng thái cho CanSm (<User\_Controller-BusOff>(), xem [SWS\_CANIF\_00014]).

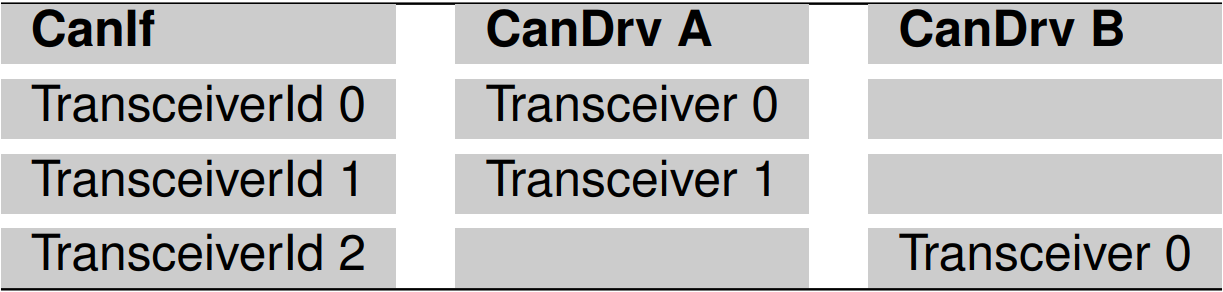
[SWS\_CANIF\_00653] [CanIf phải cung cấp một ControllerId, mà trừu tượng hóa từ các Bộ điều khiển khác nhau của các phiên bản CanDrv khác nhau. Phạm vi của ControllerIds trong CanIf phải bắt đầu với '0'. Nó phải được cấu hình thông qua CanIfCtrlId.]()

Example:

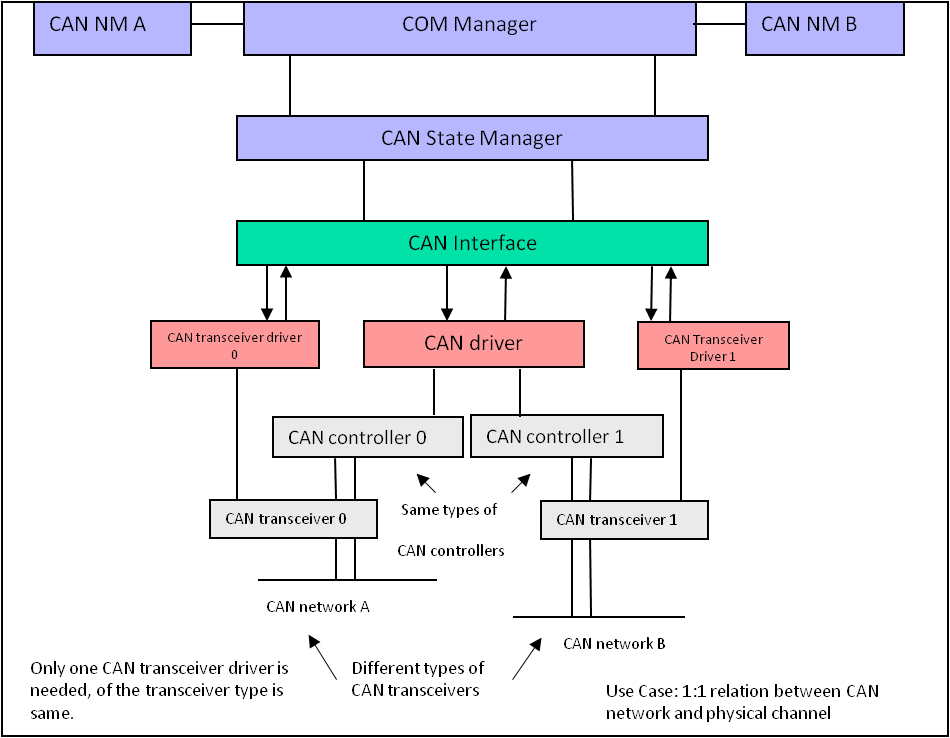


[SWS\_CANIF\_00655] [CanIf phải cung cấp một TransceiverId, mà trừu tượng hóa từ các Transceiver khác nhau của các phiên bản CanTrcv khác nhau. Phạm vi của TransceiverIds trong CanIf phải bắt đầu với '0'. Nó phải được cấu hình thông qua CanIfTrcvId.]()

Example:



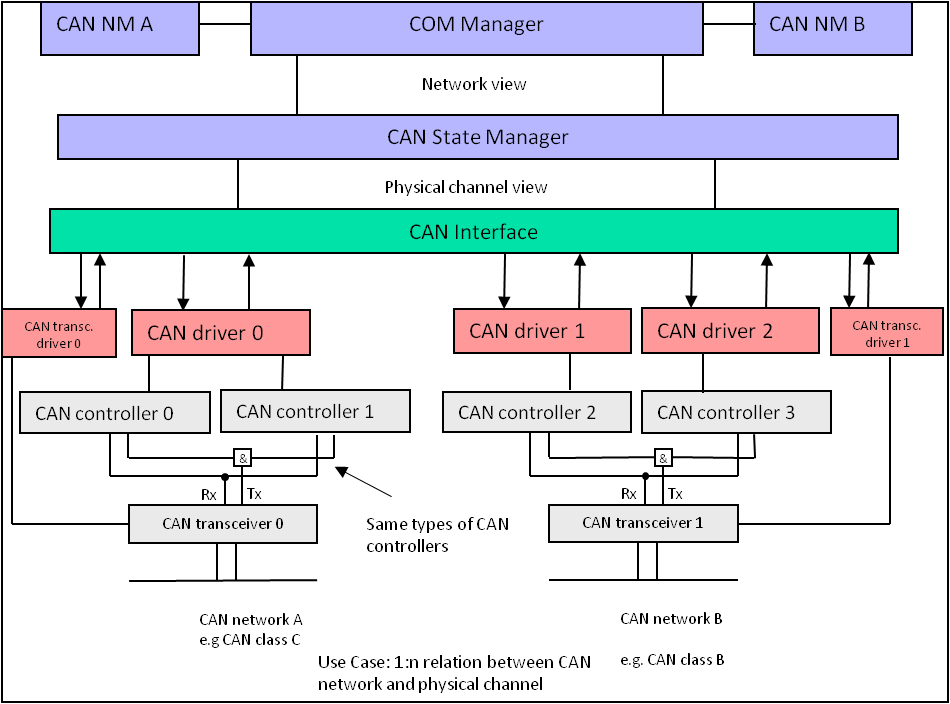
Trong quá trình thông báo, CanIf ánh xạ tham số gốc của Bộ điều khiển CAN hoặc Bộ truyền CAN từ mô-đun Driver sang CanSm. Quá trình ánh xạ này được thực hiện khi các tham số Bộ điều khiển CAN hoặc Bộ truyền CAN được tham chiếu được cấu hình với các tham số CanIf trừu tượng ControllerId hoặc TransceiverId.



**Figure 7.2: Physical channel view definition example A**

CanIf hỗ trợ nhiều kênh CAN vật lý. Các kênh này phải được phân biệt bởi CanSm để điều khiển mạng. API của CanIf cung cấp yêu cầu và kiểm soát đọc cho nhiều kênh CAN vật lý cơ bản.

Hơn nữa, CanIf không phân biệt giữa các loại đặc biệt của các lớp vật lý CAN (tức là CAN tốc độ thấp hoặc CAN tốc độ cao), mà một hoặc nhiều Bộ điều khiển CAN được kết nối đến.



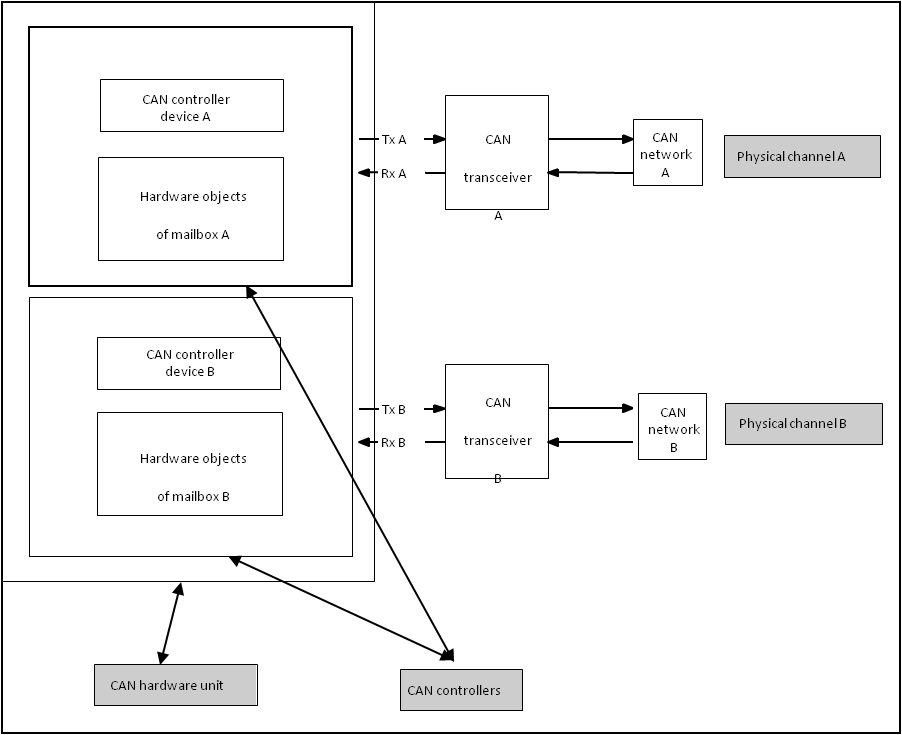
**Figure 7.3: Physical channel view definition example B**

#### CAN Hardware Unit

Đơn vị Phần cứng CAN kết hợp một hoặc nhiều mô-đun Bộ điều khiển CAN cùng loại, có thể được đặt trên chip hoặc là các thiết bị độc lập nằm ngoài. Mỗi Đơn vị Phần cứng CAN được phục vụ bởi CanDrv tương ứng.

Nếu sử dụng các loại Bộ điều khiển CAN khác nhau, cũng cần áp dụng các loại CanDrv khác nhau với một API thống nhất cho CanIf. CanIf thu thập thông tin về số lượng và loại Bộ điều khiển CAN cũng như các Đối tượng Phần cứng của chúng vào thời điểm cấu hình. Điều này cho phép truy cập đến Bộ điều khiển CAN một cách minh bạch và độc lập với phần cứng từ các mô-đun lớp trên sử dụng HOHs (xem phần 7.2 “Các khóa đối tượng phần cứng” và phần 7.24 “Hỗ trợ nhiều Bộ điều khiển CAN”).

Hình 7.4 cho thấy một Đơn vị Phần cứng CAN bao gồm hai Bộ điều khiển CAN cùng loại được kết nối với hai kênh vật lý:



**Figure 7.4:** **Typical CAN Hardware Unit**

#### BasicCAN and FullCAN reception

CanIf phân biệt giữa việc xử lý BasicCAN và FullCAN để kích hoạt việc lọc chấp nhận phần mềm.

Một hộp thư CAN (Đối tượng Phần cứng) cho hoạt động FullCAN chỉ kích hoạt việc truyền hoặc nhận một số CanIds duy nhất. Tương tự, hoạt động BasicCAN của một Đối tượng Phần cứng cho phép truyền hoặc nhận một dãy CanIds.

Một Đối tượng Nhận Phần cứng cho việc nhận BasicCAN được cấu hình có thể nhận một dãy CanIds, đi qua bộ lọc chấp nhận phần cứng của nó. Dãy này có thể vượt quá danh sách được xác định trước của các Rx L-PDU để được nhận bởi HRH này. Do đó, CanIf sau đó phải thực hiện việc lọc phần mềm để chỉ truyền qua danh sách được xác định trước của các Rx L-PDU tới các mô-đun lớp trên tương ứng. Để biết thêm chi tiết, vui lòng tham khảo phần 7.20 “Bộ lọc nhận phần mềm”.

[SWS\_CANIF\_00467] [CanIf phải cấu hình và lưu trữ một thứ tự trên HTHs và HRHs cho tất cả các HOH tạo ra từ các bộ chứa cấu hình CanIfHthCfg và CanIfHrhCfg.]()

[SWS\_CANIF\_00468] [CanIf phải tham chiếu đến một bộ lọc chấp nhận phần cứng cho mỗi HOH tạo ra từ các tham số cấu hình CanIfHthIdSymRef và CanIfHrhIdSymRef.]()

Sự khác biệt chính giữa hoạt động BasicCAN và FullCAN nằm ở nhu cầu của cơ chế lọc chấp nhận phần mềm (xem phần 7.20 “Bộ lọc nhận phần mềm”).

[SWS\_CANIF\_00469] [CanIf phải cung cấp khả năng cấu hình và lưu trữ một bộ lọc chấp nhận phần mềm cho mỗi HRH của loại BasicCAN được cấu hình bởi tham số CanIfHrhSoftwareFilter.]()

[SWS\_CANIF\_00211] [CanIf phải thực thi việc lọc chấp nhận phần mềm từ [SWS\_CANIF\_00469] cho HRH được truyền qua hàm gọi lại CanIf\_RxIndication().]()

Các đối tượng BasicCAN và FullCAN có thể tồn tại cùng một cấu hình. Nhiều Đối tượng Nhận BasicCAN và FullCAN có thể được sử dụng, nếu được cung cấp bởi các Bộ điều khiển CAN cơ bản.

[SWS\_CANIF\_00877] [Nếu CanIf nhận một L-PDU (xem CanIf\_RxIndication()), nó phải thực hiện các so sánh sau để chọn L-SDU nhận chính xác được cấu hình trong CanIfRxPduCfg:

• So sánh CanIfRxPduCanId với Mailbox->CanId được truyền (Can\_IdType) trừ hai bit có ý nghĩa nhất

• So sánh CanIfRxPduCanIdType với hai bit có ý nghĩa nhất của Mailbox->CanId (Can\_IdType)

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

Cơ bản, CanIf hỗ trợ việc nhận các ID CAN tiêu chuẩn hoặc mở rộng trên một Kênh CAN Vật lý bằng các tham số CanIfTxPduCanIdType và CanIfRxPduCanIdType.

[SWS\_CANIF\_00281] [CanIf phải chấp nhận và xử lý các ID CAN tiêu chuẩn và mở rộng trên cùng một Kênh Vật l

[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)*)*

Cơ bản, CanIf hỗ trợ việc nhận các ID CAN tiêu chuẩn hoặc mở rộng trên một Kênh CAN Vật lý thông qua các tham số CanIfTxPduCanIdType và CanIfRxPduCanIdType.

[SWS\_CANIF\_00281] [CanIf phải chấp nhận và xử lý các ID CAN tiêu chuẩn và mở rộng trên cùng một Kênh Vật lý (= hoạt động chế độ hỗn hợp).](SRS\_Can\_01140)

Trong một hoạt động chế độ hỗn hợp, các ID CAN tiêu chuẩn và mở rộng có thể được sử dụng kết hợp cùng một lúc trên cùng một mạng CAN. Hoạt động chế độ hỗn hợp có thể được thực hiện, nếu các Đối tượng CAN cơ bản/đầy đủ đã được cấu hình riêng biệt cho hoạt động StandardCAN hoặc ExtendedCAN sử dụng các tham số cấu hình CanIfTxPduCanIdType và CanIfRxPduCanIdType. Trong trường hợp hoạt động chế độ hỗn hợp, thuật toán lọc chấp nhận phần mềm (xem phần 7.20 “Bộ lọc nhận phần mềm”) phải có khả năng xử lý cả hai loại CanIds.

[SWS\_CANIF\_00281] là một tính năng tùy chọn. Tính năng này có thể được thực hiện bằng các biến thể khác nhau của triển khai, không có tùy chọn cấu hình nào có sẵn.

#### Initialization

EcuM gọi hàm CanIf\_Init() của CanIf để khởi tạo toàn bộ CanIf (xem [SWS\_CANIF\_00001]). Tất cả các biến toàn cục và cấu trúc dữ liệu được khởi tạo bao gồm các cờ và bộ đệm trong quá trình khởi tạo. EcuM thực hiện việc khởi tạo CanDrvs và CanTrcvs một cách riêng biệt thông qua việc gọi các dịch vụ khởi tạo tương ứng của chúng (tham khảo [1] và [2, Specification of CAN Transceiver Driver]).

CanIf mong đợi rằng CAN Controller sẽ giữ ở trạng thái Dừng như sau khi thiết lập lại nguồn sau khi quá trình khởi tạo đã hoàn thành. Ở chế độ này, CanIf và CanDrv không thể truyền hoặc nhận các L-PDU CAN (xem [SWS\_CANIF\_00001]).

Nếu cần thiết phải thiết lập lại toàn bộ các mô-đun CAN trong quá trình chạy, EcuM phải gọi CanSm (xem [3]) để khởi động các chuyển đổi trạng thái cần thiết của CAN Controller thông qua việc gọi dịch vụ API CanIf\_SetControllerMode() của mô-đun Giao diện CAN. CanIf ánh xạ các cuộc gọi từ CanSm thành các cuộc gọi của các CanDrvs tương ứng (xem phần con 8.6.3).

#### Transmit request

Hàm yêu cầu truyền của CanIf, CanIf\_Transmit() ([SWS\_CANIF\_00005]), là một giao diện chung cho các tầng trên để truyền các L-PDU trên mạng CAN. Các tầng giao tiếp trên chỉ khởi động việc truyền thông qua dịch vụ của CanIf mà không trực tiếp truy cập vào CanDrv. Yêu cầu Truyền được khởi động thành công nếu CanDrv có thể viết dữ liệu L-PDU vào đối tượng truyền phần cứng CAN.

Các tầng trên sử dụng dịch vụ API CanIf\_Transmit() để khởi động một yêu cầu truyền (tham khảo phần con 8.3.6 “CanIf\_Transmit”).

CanIf thực hiện các hành động sau cho việc truyền L-PDU khi gọi dịch vụ CanIf\_Transmit():

- Kiểm tra trạng thái khởi tạo của CanIf

- Xác định CanDrv (chỉ nếu có nhiều CanDrvs được sử dụng)

- Xác định HTH để truy cập vào đối tượng truyền phần cứng CAN

- Gọi Can\_Write() của CanDrv

Việc truyền được hoàn thành thành công nếu dịch vụ yêu cầu truyền CanIf\_Transmit() trả về E\_OK.

[SWS\_CANIF\_00382] [Nếu một L-PDU được yêu cầu được truyền qua một chế độ kênh PDU, mà bằng CANIF\_OFFLINE, CanIf phải báo mã lỗi chạy thời gian CANIF\_E\_STOPPED cho dịch vụ Det\_ReportRuntimeError() của DET và CanIf\_Transmit() phải trả về E\_NOT\_OK.](SRS\_Can\_01126)

Lưu ý cho [SWS\_CANIF\_00382]: Xem phần con 7.19.2 “Các chế độ kênh PDU”.

Nếu cuộc gọi của Can\_Write() trả về CAN\_BUSY, vui lòng tham khảo phần 7.12 “Xác nhận truyền” để biết thêm chi tiết.

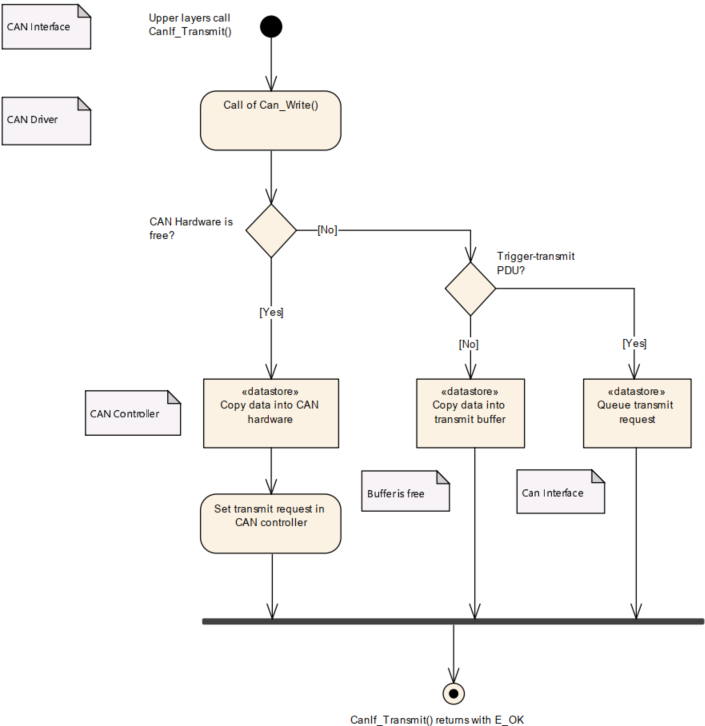
#### Transmit data flow

Hàm yêu cầu Truyền CanIf\_Transmit() dựa trên L-PDUs. Truy cập vào dữ liệu cụ thể của L-SDU được tổ chức bằng các tham số sau:

- L-PDU truyền => ID L-SDU

- Tham chiếu đến một cấu trúc dữ liệu, chứa dữ liệu liên quan đến L-SDU: Con trỏ đến L-SDU, con trỏ đến MetaData và độ dài của L-SDU.

Tham chiếu đến cấu trúc dữ liệu L-SDU được sử dụng như một tham số trong một số dịch vụ API của CanIf, ví dụ như CanIf\_Transmit() hoặc dịch vụ gọi lại <User\_RxIndica- tion()>. Trong trường hợp L-PDU được cấu hình cho việc truyền kích hoạt, con trỏ L-SDU là một con trỏ null.

****

**Figure 7.5: Transmit data flow**

CanIf lưu thông tin về các đối tượng phần cứng có sẵn được cấu hình cho mục đích truyền. Hàm CanIf\_Transmit() ánh xạ CanTxPduId tới HTH tương ứng và gọi hàm Can\_Write() (xem [SWS\_CANIF\_00318]).

[SWS\_CANIF\_00904] [Nếu Tính năng Trình chiếu Bus được kích hoạt toàn cầu (xem CanIfBusMirroringSupport) và đã được kích hoạt bằng cuộc gọi CanIf\_EnableBusMirroring() cho một CAN Controller, CanIf sẽ lưu nội dung của mỗi khung trước khi nó được truyền trên bộ điều khiển đó bằng cách sử dụng Can\_Write().]

Chú ý: Nội dung khung chỉ nên được cung cấp cho mô-đun Trình chiếu Bus khi nó thực sự được gửi đi. Do đó, nội dung phải được lưu trữ để có thể được cung cấp cho mô-đun Trình chiếu Bus từ bên trong hàm CanIf\_TxConfirmation()

.

#### Transmit buffering

##### General behavior

Tại mức CanIf, quá trình truyền bắt đầu với việc gọi hàm CanIf\_Transmit() và kết thúc với việc gọi dịch vụ callback của module tầng trên <User\_TxConfirmation()>. Trong suốt quá trình truyền, CanIf, CanDrv và CAN Mailbox cùng nhau lưu trữ L-PDU cần truyền chỉ một lần tại một vị trí duy nhất. Tùy thuộc vào phương pháp truyền, vị trí này có thể là:

- Đối tượng truyền của phần cứng CAN, hoặc

- Bộ đệm L-PDU truyền bên trong CanIf, nếu bộ đệm truyền được bật.

Đối với truyền kích hoạt, CanIf chỉ cần lưu yêu cầu truyền cho L-PDU đã cho mà không cần lưu dữ liệu của nó. Dữ liệu sẽ được lấy đúng lúc thông qua hàm truyền kích hoạt khi HTH lại sẵn sàng. Điều quan trọng cần lưu ý là một L-PDU truyền duy nhất được yêu cầu truyền không bao giờ được lưu trữ hai lần. Hành vi này phù hợp với cách giao tiếp định kỳ thông thường trên mạng CAN.

Nếu bộ đệm truyền được bật, CanIf sẽ lưu trữ một L-PDU truyền trong bộ đệm truyền L-PDU của CanIf (CanIfBufferCfg) nếu nó bị từ chối bởi CanDrv tại yêu cầu truyền.

Thực chất, bộ đệm tổng thể trong CanIf để lưu trữ các L-PDU truyền bao gồm một hoặc nhiều cấu hình CanIfBufferCfg. Mỗi CanIfBufferCfg được gán cho một hoặc nhiều tham chiếu CanIfBufferHthRef chuyên dụng và có thể được cấu hình để lưu trữ một hoặc nhiều L-PDU truyền. Tuy nhiên, như đã đề cập trước đó, chỉ một phiên bản cho mỗi L-PDU truyền có thể được lưu trữ trong tổng số CanIfBufferCfg.

Hành vi của CanIf trong quá trình truyền L-PDU thay đổi tùy thuộc vào việc có kích hoạt bộ đệm truyền hay không trong cài đặt cấu hình cho Tx L-PDU tương ứng.

Nếu bộ đệm truyền bị vô hiệu hóa:

- Nếu yêu cầu truyền đến CanDrv thất bại (ví dụ, vì hộp thư CAN Controller đang được sử dụng, BasicCAN), L-PDU không được sao chép vào hộp thư của CAN Controller và CanIf\_Transmit() trả về giá trị E\_NOT\_OK.

Nếu bộ đệm truyền được kích hoạt:

- Nếu yêu cầu truyền đến CanDrv thất bại, L-PDU có thể được lưu trữ trong một CanIfTxBuffer, tùy thuộc vào cấu hình CanIfTxBuffer. Trong trường hợp này, API CanIf\_Transmit() trả về giá trị E\_OK, mặc dù việc truyền không thể được thực hiện ngay lập tức. CanIf chịu trách nhiệm xử lý việc truyền L-PDU còn tồn đọng thông qua callback CanIf\_TxConfirmation(), và lớp trên không cần phải thử lại yêu cầu truyền.

Số lượng bộ đệm truyền CanIf Tx L-PDU có sẵn có thể được cấu hình hoàn toàn độc lập với số lượng Tx L-PDU được sử dụng được xác định trong tệp mô tả mạng CAN cho ECU này.

Theo [SWS\_CANIF\_00835], một Tx L-PDU chỉ định HTHs thông qua container cấu hình CanIfBufferCfg (xem CanIfBufferCfg). Điều này vẫn đúng ngay cả khi bộ đệm truyền không cần thiết. Trong trường hợp này, kích thước bộ đệm (xem CanIfBufferSize) của container CanIfBufferCfg phải được đặt thành 0. Sau đó container cấu hình CanIfBufferCfg chỉ được sử dụng để tham chiếu đến một HTH.

##### 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 cố gắng lưu trữ một Transmit L-PDU mới hoặc yêu cầu truyền của nó trong Bộ đệm Transmit L-PDU chỉ khi CanDrv trả về CAN\_BUSY trong quá trình gọi Can\_Write() (xem [SWS\_CANIF\_00381]).

[SWS\_CANIF\_00063] [CanIf phải hỗ trợ việc lưu trữ của một CAN L-PDU cho việc truyền BasicCAN trong CanIf, nếu tham số CanIfPublicTxBuffering (xem CanIfPublicTxBuffering) được kích hoạt.] (SRS\_Can\_01020)

[SWS\_CANIF\_00849] [Đối với các Transmit L-PDU động, cũng cần lưu trữ CanId trong CanIfTxBuffer.]()

[SWS\_CANIF\_00381] [Nếu việc lưu trữ bộ đệm truyền được kích hoạt (xem [SWS\_CANIF\_00063]) và nếu cuộc gọi của Can\_Write() cho một PDU được cấu hình để truyền trực tiếp trả về với CAN\_BUSY, CanIf sẽ kiểm tra xem có thể lưu trữ CanIf Tx L-PDU, mà được yêu cầu truyền thông qua Can\_Write(), trong một CanIfTxBuffer không.] (SRS\_Can\_01126)

Khi cuộc gọi của Can\_Write() trả về CAN\_BUSY, CanDrv đã từ chối việc truyền được yêu cầu của L-PDU (xem [1]) vì không có đối tượng phần cứng trống khả dụng vào thời điểm của yêu cầu truyền (yêu cầu Tx).

[SWS\_CANIF\_00895] [Nếu độ dài dữ liệu bị từ chối vượt quá kích thước được cấu hình, CanIf sẽ:

• lưu trữ lượng dữ liệu được cấu hình và loại bỏ phần còn lại

• và báo cáo mã lỗi thời gian chạy CANIF\_E\_DATA\_LENGTH\_MISMATCH đến dịch vụ Det\_ReportRuntimeError() của DET.]

[SWS\_CANIF\_00881] [Nếu việc lưu trữ bộ đệm truyền được kích hoạt (xem [SWS\_CANIF\_00063]) và nếu cuộc gọi của Can\_Write() cho một PDU được cấu hình để truyền theo cơ chế kích hoạt trả về với CAN\_BUSY, CanIf sẽ kiểm tra xem có thể lưu trữ Yêu cầu Truyền, đã được yêu cầu truyền thông qua Can\_Write(), trong một CanIfTxBuffer.] (SRS\_Can\_01126)

[SWS\_CANIF\_00835] [Khi CanIf kiểm tra xem có thể lưu trữ một CanIf Tx L-PDU hoặc một Yêu cầu Truyền (xem [SWS\_CANIF\_00381], [SWS\_CANIF\_00881]), điều này chỉ có thể thực hiện được nếu CanIf Tx L-PDU được gán (xem CanIfTxPduBufferRef) cho một CanIfBufferCfg (xem CanIfBuffer- Cfg), được cấu hình với kích thước bộ đệm (xem CanIfBufferSize) lớn hơn không.]()

Kích thước bộ đệm của bất kỳ CanIfTxBuffer nào chỉ có thể được cấu hình lớn hơn không, nếu lưu trữ truyền được kích hoạt. Ngoài ra, kích thước bộ đệm của một CanIfTxBuffer duy nhất chỉ có thể được cấu hình lớn hơn không nếu CanIfTxBuffer không được gán cho một FullCAN HTH (xem CanIfBufferSize).

[SWS\_CANIF\_00836] [Nếu có thể lưu trữ một CanIf Tx L-PDU hoặc một Yêu cầu Truyền, vì kích thước bộ đệm của CanIfTxBuffer được gán lớn hơn không (xem [SWS\_CANIF\_00835]), CanIf sẽ lưu trữ một CanIf Tx L-PDU hoặc Yêu cầu Truyền trong một phần tử bộ đệm trống của CanIfTxBuffer được gán, nếu CanIf Tx L-PDU hoặc Yêu cầu Truyền chưa được lưu trữ trong CanIfTxBuffer.]()

[SWS\_CANIF\_00068] [Nếu có thể lưu trữ một CanIf Tx L-PDU hoặc Yêu cầu Truyền, vì kích thước bộ đệm của CanIfTxBuffer được gán lớn hơn không (xem [SWS\_CANIF\_00835]), CanIf sẽ ghi đè lên CanIf Tx L-PDU được truyền trực tiếp trong CanIfTxBuffer được gán, nếu CanIf Tx L-PDU đã được lưu trữ trong CanIfTxBuffer khi Can\_Write() trả về CAN\_BUSY.] (SRS\_Can\_01011)

Chú ý: Không có gì để thực hiện cho các Yêu cầu Truyền đã được lưu trữ trước đó (xem [SWS\_CANIF\_00068]) do dữ liệu sẽ được CanDrv bắt trực tiếp (sử dụng CanIf\_TriggerTransmit()). Do đó, dữ liệu mới nhất sẽ được gửi tự động.

Nếu thứ tự của các yêu cầu truyền của các L-PDU khác nhau cần được giữ nguyên, các yêu cầu truyền của các module lớp trên phải được kết nối với các thông báo xác nhận truyền trước đó. Điều này có nghĩa là một L-PDU sau đó được yêu cầu truyền bởi các module lớp trên chỉ khi xác nhận truyền của L-PDU trước đó được thông báo bởi CanIf.

Chú ý: Ngoài ra, thứ tự của các yêu cầu truyền có thể khác nhau tùy thuộc vào số lượng các đối tượng truyền phần cứng được cấu hình.

[SWS\_CANIF\_00837] [Nếu kích thước bộ đệm lớn hơn không, tất cả các phần tử bộ đệm đều bận và CanIf\_Transmit() được gọi với một L-PDU mới (không có bản sao khác của cùng một L-PDU đã được lưu trữ trong bộ đệm), thì L-PDU mới hoặc Yêu cầu Truyền của nó sẽ không được lưu trữ và CanIf\_Transmit() sẽ trả về E\_NOT\_OK.]()

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

[SWS\_CANIF\_00386] [CanIf sẽ đánh giá trong quá trình xác nhận việc truyền (xem [SWS\_CANIF\_00007]), liệu có L-PDU CanIf Tx hoặc Yêu cầu Truyền đang chờ lưu trữ trong các bộ đệm CanIfTxBuffers, được gán cho Đối tượng Truyền Phần cứng mới không (xem [SWS\_CANIF\_00466]).]()

[SWS\_CANIF\_00668] [Nếu L-PDU CanIf Tx hoặc Yêu cầu Truyền đang chờ lưu trữ trong các CanIfTxBuffers theo [SWS\_CANIF\_00386], thì CanIf sẽ gọi Can\_Write() cho L-PDU CanIf Tx hoặc Yêu cầu Truyền đó (của cái được gán cho Đối tượng Truyền Phần cứng mới) với ưu tiên cao nhất (xem [SWS\_CANIF\_00070]).]()

[SWS\_CANIF\_00070] [CanIf sẽ truyền các L-PDU hoặc Yêu cầu Truyền được lưu trữ trong các bộ đệm L-PDU Truyền theo thứ tự ưu tiên (xem [12, ISO 11898-1:2015]) cho mỗi HTH. CanIf sẽ không phân biệt giữa các L-PDU và Yêu cầu Truyền.] ()

[SWS\_CANIF\_00183] [Khi CanIf gọi hàm Can\_Write() cho các L-PDU ưu tiên và Yêu cầu Truyền được lưu trữ trong CanIfTxBuffer và giá trị trả về của Can\_Write() là E\_OK, sau đó CanIf sẽ loại bỏ L-PDU hoặc Yêu cầu Truyền này khỏi Bộ đệm L-PDU Truyền ngay lập tức, trước khi xác nhận việc truyền trả về.]()

Hành vi được chỉ định trong [SWS\_CANIF\_00183] đơn giản hóa việc lựa chọn L-PDU truyền mới được lưu trữ trong Bộ đệm L-PDU Truyền.

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

[SWS\_CANIF\_00387] [Khi hàm CanIf\_Init() được gọi, CanIf sẽ khởi tạo mọi Bộ đệm L-PDU Truyền được gán cho CanIf.]()

Yêu cầu [SWS\_CANIF\_00387] là cần thiết để ngăn chặn việc truyền dữ liệu cũ sau khi khởi động lại Bộ điều khiển CAN.

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

[SWS\_CANIF\_00033] [CanIf phải bảo vệ khỏi việc truy cập song song vào Bộ đệm L-PDU Truyền để truyền L-PDU và Yêu cầu Truyền.](SRS\_Can\_- 01114)

Điều này có thể được thực hiện bằng cách sử dụng các khu vực độc quyền được xác định trong BSW Scheduler. Các khu vực độc quyền này có thể được cấu hình sao cho tất cả các ngắt sẽ bị vô hiệu hóa khi khu vực độc quyền được nhập vào. Các dịch vụ tương ứng từ mô-đun BSW Scheduler là SchM\_Enter\_CanIf() và SchM\_Exit\_CanIf().

Lý do: đối với [SWS\_CANIF\_00033]: Truy cập đa nhiệm vào Bộ đệm L-PDU Truyền không thể luôn luôn được tránh. Các truy cập Bộ đệm L-PDU Truyền như lưu trữ một L-PDU mới hoặc loại bỏ L-PDU đã được truyền có thể xảy ra theo cách chia lãnh đạo.

#### Transmit confirmation

Nếu một yêu cầu truyền trước đó được hoàn thành thành công, CanDrv thông báo nó cho CanIf

bằng cách gọi CanIf\_TxConfirmation() ([SWS\_CANIF\_00007]).

[SWS\_CANIF\_00905] [Nếu Bus Mirroring được bật toàn cầu (xem CanIfBusMirror- ingSupport) và đã được kích hoạt bằng cách gọi CanIf\_EnableBusMirroring () cho một CAN Controller, CanIf phải gọi Mirror\_ReportCanFrame() cho mỗi lần truyền khung trên bộ điều khiển đó được xác nhận bằng CanIf\_TxConfir- mation(), cung cấp nội dung đã lưu trữ và ID CAN thực tế.] (SRS\_Can\_01172)

[SWS\_CANIF\_00383] [Khi thông báo gọi lại CanIf\_TxConfirma- tion() được gọi, CanIf phải xác định lớp giao tiếp lớp trên cùng (xem [SWS\_CANIF\_00414]), mà được liên kết với L-PDU đã được truyền thành công, và phải thông báo cho nó về quá trình truyền bằng cách gọi dịch vụ xác nhận truyền của CanIf <User\_TxConfirmation>(E\_OK).]()

Lưu ý cho [SWS\_CANIF\_00383]: Xem phần 7.12 “Xác nhận truyền”.

Dịch vụ gọi lại <User\_TxConfirmation>() được thực hiện bởi mô-đun lớp trên cùng được thông báo.

Một mô-đun lớp giao tiếp trên cùng có thể được thiết kế hoặc cấu hình sao cho các xác nhận truyền có thể được xử lý với các dịch vụ gọi lại đơn lẻ hoặc nhiều dịch vụ gọi lại cho các L-PDU khác nhau hoặc các nhóm L-PDU. Tất cả các dịch vụ đó được gọi bởi CanIf khi xác nhận truyền của yêu cầu truyền L-PDU tương ứng. L-PDU Truyền cho phép phân phát các dịch vụ xác nhận khác nhau được liên kết với mô-đun lớp trên cùng mục tiêu. Sự phân công này được thực hiện tĩnh trong quá trình cấu hình.

Một L-PDU truyền chỉ có thể được phân cho một dịch vụ gọi lại xác nhận truyền đơn lẻ. Vui lòng xem phần con-điều hướng 8.6.3.2 “<User\_TxConfirmation>”.

[SWS\_CANIF\_00740] [Nếu CanIfPublicTxConfirmPollingSupport được bật, CanIf phải lưu trữ thông tin về một xác nhận Tx nhận được cho mỗi CAN Con- troller, nếu trạng thái chế độ điều khiển của bộ điều khiển đó ở trạng thái CAN\_CS\_STARTED.]()

#### Receive data flow

Theo Kiến trúc Phần mềm Cơ bản AUTOSAR, dữ liệu nhận được sẽ được đánh giá và xử lý trong các ngăn xếp truyền thông tầng trên (tức là AUTOSAR COM, CanNm, CanTp, DCM). Điều này có nghĩa là các module tầng trên không được phép làm việc với (tức là thay đổi) các bộ đệm của CanDrv (Rx) và cũng không có quyền truy cập vào các bộ đệm của CanIf (Tx).

CanIf cung cấp bộ đệm nội bộ trong đường nhận chỉ khi CanIfPublicReadRxPduDataApi được đặt thành TRUE (tham khảo mục 7.15). Bộ đệm truyền (Tx buffering) được đề cập trong mục 7.11 và các L-PDU động được đề cập trong mục 7.4.

Khi có một L-PDU mới được nhận, CanDrv sẽ gọi hàm CanIf\_RxIndication() (tham khảo [SWS\_CANIF\_00006]) của CanIf. Việc truy cập vào dữ liệu cụ thể của L-PDU được tổ chức bởi các tham số sau:

- \*\*Hardware Receive Handle (HRH)\*\*

- \*\*Received CAN Identifier (CanId)\*\*

- \*\*Received Data Length\*\*

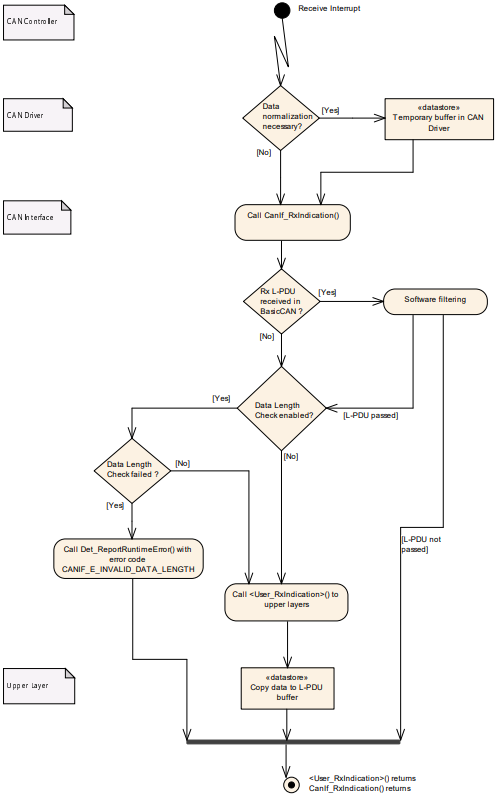
- \*\*Reference to Received L-PDU\*\*

L-PDU nhận được phụ thuộc vào phần cứng (thứ tự nibble và byte, loại truy cập) và được phân bổ cho tầng thấp nhất trong hệ thống truyền thông - CanDrv. HRH phục vụ như một liên kết giữa CanDrv và module tầng trên sử dụng L-PDU. HRH xác định một đối tượng nhận phần cứng CAN, nơi một L-PDU CAN mới đã được nhận.

Sau khi CanDrv chỉ ra rằng một L-PDU đã được nhận (gọi hàm CanIf\_RxIndication()), CanIf sẽ tiếp tục xử lý như mô tả trong mục 7.14 về chỉ thị nhận. CanIf không thể nhận ra liệu CanDrv sử dụng bộ đệm tạm thời hay truy cập trực tiếp vào phần cứng. Nó mong đợi dữ liệu L-PDU đã được chuẩn hóa trong các lời gọi của CanIf\_RxIndication().

Đối tượng nhận CAN phần cứng sẽ bị khóa cho đến khi kết thúc quá trình sao chép sang bộ đệm tạm thời hoặc bộ đệm của module tầng trên. Đối tượng phần cứng sẽ được giải phóng ngay lập tức sau khi CanIf\_RxIndication() của CanIf trả về để tránh mất dữ liệu.

CanDrv, CanIf và module tầng trên, nơi thuộc về L-PDU nhận được, truy cập cùng một bộ đệm trung gian tạm thời, bộ đệm này có thể nằm trong đối tượng nhận phần cứng của Bộ điều khiển CAN hoặc là bộ đệm tạm thời trong CanDrv.



**Figure 7.6: Receive data flow**

#### Receive indication

Khi gọi hàm CanIf\_RxIndication() (xem [SWS\_CANIF\_00006]), các tham số của nó tham chiếu đến một CAN L-PDU mới nhận. Khi hàm CanIf\_RxIndication() được gọi, CanIf sẽ đánh giá CAN L-PDU để chấp nhận và chuẩn bị L-SDU để sau này có thể truy cập bởi các tầng giao tiếp trên. CanIf sẽ thông báo cho các module tầng trên về sự kiện không đồng bộ này bằng cách sử dụng <User\_RxIndication>() (xem tiểu mục 8.6.3.3 "<User\_RxIndication>", [SWS\_CANIF\_00012]), nếu được cấu hình và nếu CAN L-PDU này được phát hiện và chấp nhận thành công để xử lý tiếp. Các yêu cầu chi tiết cho hành vi này được nêu ra dưới đây.

[SWS\_CANIF\_00906] [Nếu Hỗ trợ Bus Mirroring được bật toàn cầu (xem CanIfBusMirroringSupport) và đã được kích hoạt bằng cách gọi CanIf\_EnableBusMirroring() cho một Bộ điều khiển CAN, CanIf sẽ gọi Mirror\_ReportCanFrame() cho mỗi khung nhận trên bộ điều khiển đó được chỉ định với CanIf\_RxIndication().](SRS\_Can\_01172)

[SWS\_CANIF\_00389] [Nếu hàm CanIf\_RxIndication() được gọi, CanIf sẽ xử lý Bộ lọc phần mềm trên L-PDU nhận được, nếu được cấu hình (xem số lượng của CanIfHrhRangeCfg bằng 0..\*). Nếu Bộ lọc phần mềm từ chối L-PDU nhận được, CanIf sẽ kết thúc chỉ thị nhận cho lần gọi CanIf\_RxIndication() đó.]()

Ghi chú cho [SWS\_CANIF\_00389]: Xem mục 7.20.

[SWS\_CANIF\_00390] [Nếu CanIf chấp nhận một L-PDU nhận qua CanIf\_RxIndication() trong quá trình Bộ lọc phần mềm (xem [SWS\_CANIF\_00389]), CanIf sẽ xử lý kiểm tra Độ dài dữ liệu sau đó, nếu được cấu hình (xem CanIfPrivateDataLengthCheck và CanIfRxPduDataLengthCheck).]()

Để biết thêm chi tiết, vui lòng tham khảo mục 7.21 "Kiểm tra Độ dài Dữ liệu".

[SWS\_CANIF\_00297] [Nếu CanIf đã chấp nhận một L-PDU được nhận thông qua CanIf\_RxIndication() trong quá trình Kiểm tra Độ dài Dữ liệu (xem [SWS\_CANIF\_00390]), CanIf sẽ sao chép số byte tương ứng với Độ dài Dữ liệu được cấu hình (xem ECUC\_CanIf\_00599) vào bộ đệm nhận tĩnh, nếu được cấu hình cho L-PDU đó (xem [SWS\_CANIF\_00198], ECUC\_CanIf\_00600).]()

[SWS\_CANIF\_00851] [Nếu MetaData được cấu hình cho một L-SDU được nhận, CanIf sẽ sao chép dữ liệu PDU vào bộ đệm nhận tĩnh và CAN ID vào MetaDataItem kiểu CAN\_ID\_32.]()

[SWS\_CANIF\_00056] [Nếu CanIf chấp nhận một L-PDU nhận được qua CanIf\_RxIndication() trong quá trình Kiểm tra Độ dài Dữ liệu (xem [SWS\_CANIF\_00390], [SWS\_CANIF\_00026]), CanIf sẽ xác định xem một module tầng trên đích đã được cấu hình (xem mô tả cấu hình của [SWS\_CANIF\_00012], CanIfRxPdu- UserRxIndicationUL, CanIfRxPduUserRxIndicationName) để được gọi với dịch vụ chỉ thị nhận của nó cho L-SDU nhận được hay không.]()

[SWS\_CANIF\_00135] [Nếu một module tầng trên đích đã được cấu hình để được gọi với dịch vụ chỉ thị nhận của nó (xem [SWS\_CANIF\_00056]), CanIf sẽ gọi dịch vụ gọi lại chỉ thị nhận được được cấu hình này (xem CanIfRxPduUserRxIndi- cationName) và cung cấp các tham số cần thiết cho các hàm gọi lại thông báo của tầng trên (xem [SWS\_CANIF\_00012]) dựa trên các tham số của CanIf\_- RxIndication().]()

Ghi chú: Một L-PDU nhận duy nhất chỉ có thể được gán cho một dịch vụ gọi lại chỉ thị nhận duy nhất (tham khảo số lượng của CanIfRxPduUserRxIndicationName).

Tổng quan: CanIf thực hiện các bước sau tại một cuộc gọi của CanIf\_RxIndication():

- Bộ lọc phần mềm (chỉ BasicCAN), nếu được cấu hình

- Kiểm tra Độ dài Dữ liệu, nếu được cấu hình

- Sao chép L-SDU nhận được vào bộ đệm, nếu được cấu hình

- Gọi dịch vụ gọi lại chỉ thị nhận của tầng trên, nếu được cấu hình.

#### Read received data

Hàm giao diện chung CanIf\_ReadRxPduData() (xem [SWS\_CANIF\_00194]) là một giao diện thông thường cho các module tầng trên đọc các L-SDU CAN mới nhận được từ mạng CAN. Các module tầng trên khởi tạo yêu cầu nhận chỉ thông qua dịch vụ CanIf mà không cần truy cập trực tiếp vào CanDrv. Yêu cầu nhận được khởi tạo sẽ được hoàn thành thành công nếu CanIf đã ghi L-SDU nhận được vào bộ đệm I-PDU của module tầng trên.

Hàm CanIf\_ReadRxPduData() cho phép đọc dữ liệu mà không phụ thuộc vào sự kiện nhận (RxIndication). Khi nó được kích hoạt vào thời gian cấu hình (xem CanIfPublicReadRxPduDataApi), không nhất thiết phải cấu hình một dịch vụ chỉ thị nhận cho cùng một L-SDU (xem CanIfRxPduUserRxIndicationUL). Nếu cần, chỉ thị nhận có thể được kích hoạt, nếu muốn.

Theo cách này, loại cơ chế để nhận L-SDU (trong các module tầng trên của CanIf) có thể được chọn vào thời gian cấu hình bằng tham số CanIfRxPduUser- RxIndicationUL và tham số CanIfRxPduReadData theo nhu cầu của module tầng trên, mà L-SDU nhận tương ứng thuộc về. Để biết chi tiết, vui lòng tham khảo mục 9.9 “Đọc dữ liệu nhận được”.

[SWS\_CANIF\_00198] [Nếu tham số cấu hình CanIfPublicReadRxPdu- DataApi được đặt thành TRUE, CanIf sẽ lưu trữ mỗi L-SDU nhận được, mà CanI- fRxPduReadData được kích hoạt, vào một bộ đệm L-SDU nhận được. Điều này có nghĩa là nếu tham số cấu hình CanIfRxPduReadData được đặt thành TRUE, CanIf phải cấp phát một bộ đệm L-SDU nhận được cho L-SDU nhận được này.]

[SWS\_CANIF\_00199] [Sau cuộc gọi CanIf\_RxIndication() và qua quá trình lọc phần mềm và Kiểm tra Độ dài Dữ liệu, CanIf sẽ lưu trữ L-SDU nhận được vào bộ đệm L-SDU nhận được này. Trong quá trình cuộc gọi CanIf\_ReadRxPduData(), bộ đệm L-SDU nhận được được gán chứa một L-SDU nhận được gần đây, CanIf sẽ tránh các sự kiện truy cập trước đó vào bộ đệm L-SDU nhận được đó.]

#### Read Tx/Rx notification status

Ngoài các hàm gọi lại thông báo, CanIf cung cấp dịch vụ API CanIf\_ReadTxNotifStatus() (xem [SWS\_CANIF\_00202]) để đọc trạng thái xác nhận truyền của bất kỳ L-SDU truyền nào và dịch vụ API CanIf\_ReadRxNotifStatus() được cung cấp để đọc trạng thái chỉ thị nhận của bất kỳ L-SDU nhận nào.

Các dịch vụ API của CanIf CanIf\_ReadTxNotifStatus() (xem [SWS\_CANIF\_00202]) và CanIf\_ReadRxNotifStatus() (xem [SWS\_CANIF\_00230]) có thể được kích hoạt/tắt toàn cầu hoặc cho mỗi L-SDU tại thời gian cấu hình trước biên dịch bằng cách sử dụng các tham số cấu hình CanIfPublicReadTxPduNotifyStatusApi, CanIfPubli- cReadRxPduNotifyStatusApi, CanIfTxPduReadNotifyStatus, và CanI- fRxPduReadNotifyStatus.

[SWS\_CANIF\_00472] [Nếu tham số cấu hình CanIfPublicReadTxPduNoti- fyStatusApi được đặt thành TRUE, CanIf sẽ lưu trữ trạng thái thông báo hiện tại cho mỗi L-SDU truyền.]()

[SWS\_CANIF\_00473] [Nếu tham số cấu hình CanIfPublicReadRxPduNoti- fyStatusApi được đặt thành TRUE, CanIf sẽ lưu trữ trạng thái thông báo hiện tại cho mỗi L-SDU nhận.]()

Lý do cho [SWS\_CANIF\_00391] và [SWS\_CANIF\_00393] tương ứng là [SWS\_CANIF\_00392] và [SWS\_CANIF\_00394]: Hành vi "đọc và tiêu thụ" này đảm bảo rằng ít nhất một sự kiện truyền hoặc nhận thành công đã xảy ra sau lần gọi cuối cùng của dịch vụ này.

#### Data integrity

[SWS\_CANIF\_00064] Mã chia sẻ phải là có khả năng tái nhập [CanIf phải bảo vệ các sự kiện can thiệp, có thể truy cập vào các nguồn tài nguyên chia sẻ, có thể được thay đổi trong quá trình xử lý sự kiện của CanIf, chống lại nhau.]()

Lý do: Việc cập nhật dữ liệu trong các bộ đệm của module tầng trên cũng như trong các bộ đệm nội bộ của CanIf phải được thực hiện với sự tôn trọng đến các thay đổi có thể xảy ra trong ngữ cảnh của một dịch vụ ngắt hoặc các sự kiện can thiệp khác. Các sự kiện can thiệp có thể xảy ra từ các tác vụ can thiệp, nhiều ngắt CAN, nếu nhiều kênh vật lý, ví dụ như cho các cổng, được sử dụng, hoặc trong trường hợp của các ngắt thiết bị hoặc hệ thống mạng khác, có nhu cầu truyền và nhận L-PDU trên mạng.

Xử lý các bộ đệm L-PDU/L-SDU truyền và nhận chia sẻ là vấn đề quan trọng đối với việc triển khai CanIf. Do đó, CanIf phải đảm bảo tính toàn vẹn dữ liệu và do đó sử dụng các cơ chế phù hợp để truy cập vào các nguồn tài nguyên chia sẻ như bộ đệm L-PDU/L-SDU truyền/nhận. Các sự kiện can thiệp, ví dụ như sự kiện truyền và nhận từ các Bộ điều khiển CAN khác có thể làm nguy hiểm đến tính toàn vẹn dữ liệu bằng cách ghi vào cùng một bộ đệm L-PDU/L-SDU.

Ví dụ, CanIf có thể sử dụng các dịch vụ CanDrv để kích hoạt (Can\_EnableControllerInter- rupts()) và vô hiệu hóa (Can\_Disable-ControllerInterrupts()) ngắt CAN và các thông báo của nó tại điểm vào và ra của các phần ch critical riêng biệt cho mỗi Bộ điều khiển CAN. Nếu có các nguồn tài nguyên chung cho nhiều Bộ điều khiển CAN, toàn bộ các Ngắt CAN phải bị khóa. Các phần này không được phép mất nhiều thời gian để tránh sự suy giảm nghiêm trọng về hiệu suất. Do đó, việc sao chép dữ liệu, thay đổi biến tĩnh, bộ đếm và semaphore nên được thực hiện trong các phần quan trọng này. Việc đảm bảo tính toàn vẹn dữ liệu, khả năng ngắt và tái nhập là trách nhiệm của triển khai.

API yêu cầu truyền CanIf\_Transmit() phải có khả năng hoạt động tái nhập để cho phép nhiều cuộc gọi yêu cầu truyền được gây ra bởi các sự kiện can thiệp khác nhau của các L-PDU/L-SDU khác nhau. API yêu cầu truyền của CanDrv Can\_Write() cũng hoạt động tái nhập.

#### CAN Controller Mode

##### General Functionality

CanIf cung cấp các dịch vụ để kiểm soát chế độ giao tiếp của tất cả các Bộ điều khiển CAN được hỗ trợ được đại diện bởi CanDrv. Điều này có nghĩa là tất cả các Bộ điều khiển CAN được kiểm soát bằng các dịch vụ API tương ứng được cung cấp để yêu cầu và đọc chế độ điều khiển hiện tại.

Trạng thái của Bộ điều khiển CAN có thể được thay đổi theo yêu cầu của tầng trên thông qua việc gọi dịch vụ CanIf\_SetControllerMode(). Yêu cầu được chuyển tiếp bởi CanIf thông qua API CanDrv tới Bộ điều khiển CAN được chỉ định.

Quản lý nhất quán của tất cả các Bộ điều khiển CAN kết nối trên một mạng CAN là nhiệm vụ của CanSm. Bằng cách này, CanSm chịu trách nhiệm thiết lập tất cả các Bộ điều khiển CAN của một mạng CAN theo thứ tự vào chế độ ngủ hoặc đánh thức chúng.

CanIf chấp nhận mọi yêu cầu chuyển trạng thái bằng cách gọi hàm CanIf\_SetControllerMode() hoặc CanIf\_ControllerBusOff(). CanIf không quyết định xem một chuyển đổi trạng thái yêu cầu của Bộ điều khiển CAN có hợp lệ hay không. CanIf chỉ tương tác với CanDrv bằng cách lấy chế độ hiện tại và thực thi các chuyển đổi trạng thái được yêu cầu.

Máy trạng thái liên quan đến mạng này được triển khai trong CanSm. Tham khảo [3]. CanIf chỉ lưu trữ chế độ được yêu cầu và thực hiện chuyển đổi được yêu cầu.

Gợi ý: Như một tối ưu hóa để tránh các yêu cầu thường xuyên đến CanDrv cho việc sử dụng nội bộ, trạng thái cuối cùng được chỉ ra bởi CanIf\_ControllerModeIndication() và Can\_GetControllerMode() có thể được lưu trữ cho mỗi bộ điều khiển.

Gợi ý: Phải xem xét rằng không chỉ CanSm mới có thể yêu cầu thay đổi Chế độ Bộ điều khiển CAN.

##### CAN Controller Operation Modes

The CanSm forwards requests to CanDrvs according to the requested operation mode.

[SWS\_CANIF\_00677] [Nếu một chế độ điều khiển được tham chiếu bởi ControllerId đang ở trạng thái CAN\_CS\_STOPPED và nếu tham số PduIdType trong một cuộc gọi của CanIf\_Transmit() được gán cho Bộ điều khiển CAN đó, thì cuộc gọi của CanIf\_Transmit() sẽ không dẫn đến một cuộc gọi của Can\_Write() (xem [SWS\_CANIF\_00317]) và trả về E\_NOT\_OK.] ()

[SWS\_CANIF\_00485] [Nếu một chế độ điều khiển được tham chiếu bởi ControllerId nhập vào trạng thái CAN\_CS\_STOPPED, thì CanIf phải xóa các bộ đệm truyền của CanIf được gán cho Bộ điều khiển CAN tương ứng.] ()

[SWS\_CANIF\_00739] [Nếu một chế độ điều khiển được tham chiếu bởi ControllerId nhập vào trạng thái CAN\_CS\_STOPPED, thì CanIf phải thông báo cho các tầng trên tương ứng về việc truyền thất bại bằng cách gọi <User\_TxConfirmation>(id, E\_NOT\_OK) cho mỗi TxConfirmation đang chờ gán cho Bộ điều khiển CAN đó. Nếu CanIfPublicTxConfirmPollingSupport được kích hoạt, CanIf cũng phải xóa thông tin về một TxConfirmation (xem [SWS\_CANIF\_00740]).]()

Ghi chú: Điều này đảm bảo rằng cho mỗi PDU, mà muốn được truyền qua CanIf\_Transmit(), hoặc một <User\_TxConfirmation>() tích cực hoặc tiêu cực sẽ được gọi.

[SWS\_CANIF\_00724] [Khi callback CanIf\_ControllerBusOff(Control- lerId) được gọi, CanIf phải gọi CanSM\_ControllerBusOff(Control- lerId) của CanSm hoặc một CDD (xem [SWS\_CANIF\_00559], [SWS\_CANIF\_00560]).] ()

Ghi chú cho [SWS\_CANIF\_00724]: Xem phụ mục 8.6.3.9 “<User\_ControllerMod- eIndication>”.

[SWS\_CANIF\_00711] [Khi callback CanIf\_ControllerModeIndication (ControllerId, ControllerMode) được gọi, CanIf phải gọi CanSm\_Con- trollerModeIndication(ControllerId, ControllerMode) của CanSm hoặc một CDD (xem [SWS\_CANIF\_00691], [SWS\_CANIF\_00692]).] ()

Ghi chú cho [SWS\_CANIF\_00711]: Xem phụ mục 8.6.3.9 “<User\_ControllerMod- eIndication>”.

[SWS\_CANIF\_00712] [Khi callback CanIf\_TrcvModeIndication (Transceiver, TransceiverMode) được gọi, CanIf phải gọi CanSM\_- TransceiverModeIndication(TransceiverId, TransceiverMode) của CanSm hoặc một CDD (xem [SWS\_CANIF\_00697], [SWS\_CANIF\_00698]).] ()

Ghi chú cho [SWS\_CANIF\_00712]: Xem phụ mục 8.6.3.9 “<User\_ControllerMod- eIndication>”.

##### Controller Mode Transitions

API cho các yêu cầu thay đổi trạng thái của CAN Controller hoạt động theo cách bất đồng bộ với thông báo bất đồng bộ thông qua các dịch vụ gọi lại.

Chuyển đổi thực sự sang chế độ yêu cầu xảy ra không đồng bộ dựa trên việc thiết lập các yêu cầu chuyển đổi trong phần cứng của bộ điều khiển CAN, ví dụ: yêu cầu chuyển đổi sang chế độ ngủ CAN\_CS\_SLEEP. Sau khi chuyển đổi thành công sang chế độ ví dụ: chế độ CAN\_CS\_SLEEP, CanDrv gọi hàm CanIf\_ControllerModeIndication() và CanIf lần lượt gọi hàm <User\_ControllerModeIndication>(). Nếu CAN chuyển đổi rất nhanh, CanIf\_- ControllerModeIndication() có thể được gọi trong quá trình CanIf\_SetController- Mode(). Điều này phụ thuộc vào cách triển khai cụ thể.

Các chuyển đổi chế độ không thành công hoặc không có của các CAN Controller phải được theo dõi bởi các module tầng trên. Các chuyển đổi chế độ CAN\_CS\_STARTED và CAN\_CS\_STOPPED được xử lý tương tự nhau.

Các module tầng trên của CanIf có thể kiểm tra chế độ Controller hiện tại bằng CanIf\_Get- ControllerMode().

Không phải tất cả các loại CAN Controller đều hỗ trợ Chế độ Ngủ và Thức dậy. Các chế độ này sau đó được bao gồm bởi CanDrv bằng cách cung cấp các chế độ hoạt động không phụ thuộc vào phần cứng qua giao diện của nó, mà phải được quản lý bởi CanIf.

Ghi chú: Có thể xảy ra trong quá trình chuyển đổi từ CAN\_CS\_STOPPED sang CAN\_CS\_SLEEP, CAN Controller có thể chỉ ra một ngắt thức dậy cho Mã tích hợp ECU.

CanIf phân biệt giữa yêu cầu thức dậy của bộ điều khiển CAN được khởi tạo từ bên trong (yêu cầu nội bộ) và yêu cầu thức dậy của mạng (yêu cầu bên ngoài). Yêu cầu nội bộ được khởi tạo bằng cách gọi hàm CanIf’s function CanIf\_SetControllerMode(ControllerId, CAN\_CS\_STARTED) và nó là một yêu cầu bất đồng bộ nội bộ. Yêu cầu bên ngoài là một sự kiện của bộ điều khiển CAN, được thông báo bởi CanDrv hoặc CanTrcv đến Mã tích hợp ECU. Để biết thông tin chi tiết, xem sơ đồ UML tương ứng trong chương "Các Trình Tỉnh Thức CAN" của tài liệu [13].

##### Wake-up

ECU hỗ trợ tỉnh dậy qua mạng CAN, bất kể phương pháp tỉnh dậy được sử dụng (trực tiếp qua bộ điều khiển CAN hoặc bộ điều khiển CAN Transceiver), chỉ khi CAN Controller và CAN Transceiver được thiết lập vào một loại chế độ "nghe cho tỉnh dậy" nào đó. Thông thường đây là Chế độ Ngủ, nơi giao tiếp thông thường bị vô hiệu hóa. Chỉ chế độ này đảm bảo rằng CAN Controller đã dừng. Do đó, ngắt tỉnh dậy có thể được kích hoạt.

###### Wake-up detection

Nếu hỗ trợ tỉnh dậy được kích hoạt (xem ECUC\_CanIf\_00843), CanIf sẽ được thông báo bởi Mã tích hợp về một sự tỉnh dậy CAN phát hiện thông qua dịch vụ CanIf\_CheckWakeup() (xem Các chuỗi tỉnh dậy CAN của [13]).

Trong trường hợp của một sự kiện "tỉnh dậy" trên bus CAN, hàm CanIf\_CheckWakeup(WakeupSource) có thể được gọi trong quá trình thực thi của EcuM\_CheckWakeup(WakeupSource) (xem các biểu đồ chuỗi tỉnh dậy của EcuM). CanIf sau đó kiểm tra thông qua tham chiếu đầu vào được cấu hình tới EcuMWakeupSource trong CanDrvs, những CanDrvs nào cần được kiểm tra. CanIf nhận thông tin này qua tham chiếu CanIfCtrlCanCtrlRef.

Dịch vụ Giao tiếp, được gọi, thuộc về dịch vụ được xác định trong quá trình cấu hình (xem CanIfDispatchCfg). Như vậy, EcuM cũng như CanSm có thể thay đổi Trạng thái CAN Controller và kiểm soát hành vi của hệ thống liên quan đến quy trình phục hồi BusOff hoặc tỉnh dậy.

[SWS\_CANIF\_00395] [Khi CanIf\_CheckWakeup(EcuM\_WakeupSourceType WakeupSource) được gọi, CanIf phải truy vấn CanDrvs / CanTrcvs qua CanTrcv\_- CheckWakeup() hoặc Can\_CheckWakeup(), xác định thiết bị phần cứng CAN cụ thể nào đã gây ra sự tỉnh dậy trên bus.]()

Lưu ý: Điều này phụ thuộc vào cách triển khai, những bộ điều khiển và bộ khuếch đại tương thích được truy vấn.

CanIf chỉ cần phải xác định thiết bị phần cứng CAN cụ thể.

[SWS\_CANIF\_00720] [Nếu ít nhất một cuộc gọi hàm của Can\_CheckWakeup() hoặc CanTrcv\_CheckWakeup() trả về E\_OK cho CanIf, thì CanIf\_CheckWakeup() phải trả về E\_OK.]()

[SWS\_CANIF\_00678] [Nếu tất cả các cuộc gọi của Can\_CheckWakeup() hoặc CanTrcv\_CheckWakeup() trả về E\_NOT\_OK cho CanIf, thì CanIf\_CheckWakeup() phải trả về E\_NOT\_OK.]()

###### Wake-up Validation

Ghi chú: Khi một CAN Controller / CAN Transceiver phát hiện một sự kiện tỉnh dậy trên bus, thì điều này sẽ được thông báo đến Bộ quản lý Trạng thái ECU trực tiếp. Nếu một sự kiện tỉnh dậy như vậy cần được xác nhận, thì EcuM (hoặc một CDD) sẽ bật CAN Controller tương ứng (CanIf\_SetControllerMode()) và CAN Transceiver (CanIf\_SetTrcvMode()) (Xem thêm chi tiết trong chương 9 của [13]).

Chú ý: CanIf thông báo cho các module lớp trên về các thông điệp nhận được sau khi Chế độ Kênh PDU đã được đặt thành CANIF\_ONLINE hoặc CANIF\_TX\_OFFLINE. Do đó, cần phải đảm bảo rằng Chế độ Kênh PDU không được đặt thành CANIF\_ONLINE hoặc CANIF\_TX\_OFFLINE nếu cần phải xác nhận tỉnh dậy.

Ghi chú: Theo [SWS\_CAN\_00411] và Sơ đồ Trạng thái CAN Controller (xem [1]), một chuyển đổi trực tiếp từ chế độ CAN\_CS\_SLEEP sang CAN\_CS\_STARTED không được phép.

[SWS\_CANIF\_00286] [Nếu CanIfPublicWakeupCheckValidSupport bằng TRUE, CanIf kích hoạt việc phát hiện cho xác nhận tỉnh dậy CAN. Do đó, CanIf lưu trữ sự kiện của cuộc gọi đầu tiên hợp lệ của CanIf\_RxIndication() của một CAN Controller đã được đặt thành CAN\_CS\_STARTED. Cuộc gọi đầu tiên của CanIf\_RxIndication() là hợp lệ:

• chỉ cho các thông điệp NM nhận được nếu CanIfPublicWakeupCheckValidByNM là TRUE

• cho tất cả các thông điệp nhận được tương ứng với một Rx PDU được cấu hình nếu CanIfPublicWakeupCheckValidByNM là FALSE.

](SRS\_Can\_01151)

[SWS\_CANIF\_00179] [<User\_ValidateWakeupEvent>(sources) sẽ được gọi trong quá trình CanIf\_CheckValidation(WakeupSource), trong đó sources được đặt thành WakeupSource, nếu sự kiện của cuộc gọi đầu tiên của CanIf\_RxIndication() được lưu trữ trong CanIf tại CAN Controller tương ứng.](SRS\_Can\_01136)

Ghi chú: Tham số của hàm <User\_ValidateWakeupEvent>() có kiểu:

• sources: EcuM\_WakeupSourceType (xem [13])

[SWS\_CANIF\_00756] [Khi chế độ điều khiển được đặt thành CAN\_CS\_SLEEP, sự kiện đã được lưu trữ từ tỉnh dậy trước đó (cuộc gọi đầu tiên của CanIf\_RxIndication) sẽ được xóa (xem [SWS\_CANIF\_00179]).]()

#### PDU channel mode control

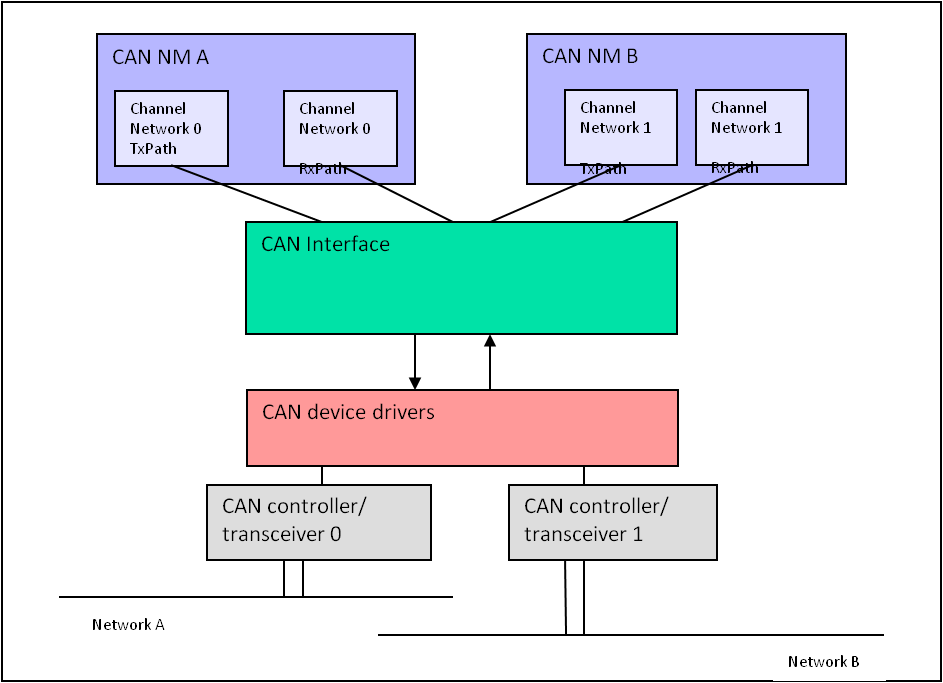
##### PDU channel groups

Mỗi L-PDU được gán cho một kênh CAN vật lý riêng biệt được kết nối với một CAN Controller và một mạng CAN. Bằng cách này, tất cả các L-PDU thuộc về một Kênh Vật lý có thể được kiểm soát trong quan điểm xử lý các nhóm kênh L-PDU logic. Những nhóm logic này đại diện cho tất cả các L-PDU của một ECU được kết nối với một mạng CAN dưới lying.

Hình 7.7 dưới đây cho thấy một cách sử dụng có thể của nhóm kênh L-PDU và mối quan hệ của nó với các lớp trên và/hoặc các mạng.

Một L-PDU chỉ có thể được gán cho một nhóm kênh.

Người dùng điển hình như PduR hoặc Quản lý Mạng chịu trách nhiệm điều khiển các chế độ hoạt động của PDU.



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

##### PDU channel modes

CanIf cung cấp các dịch vụ CanIf\_SetPduMode() và CanIf\_GetPduMode() để ngăn chặn việc xử lý:

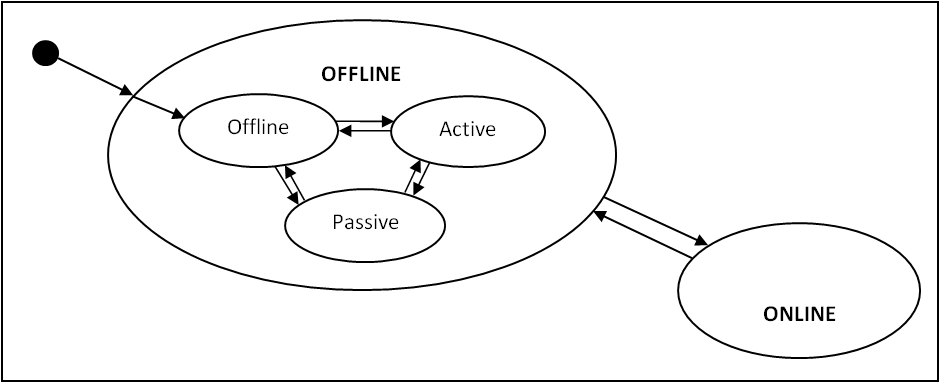
• tất cả các L-PDU truyền thuộc về một kênh logic,

• tất cả các L-PDU truyền và L-PDU nhận thuộc về một kênh logic.

Thay đổi chế độ kênh PDU chỉ được phép trong trường hợp chế độ bộ điều khiển tương ứng bằng CAN\_CS\_STARTED (xem [SWS\_CANIF\_00874]).

Trong khi CANIF\_ONLINE và CANIF\_OFFLINE ảnh hưởng đến toàn bộ giao tiếp, các chế độ kênh PDU CANIF\_TX\_OFFLINE và CANIF\_TX\_OFFLINE\_ACTIVE cho phép/tắt đường truyền truyền riêng lẻ.

CanIf cung cấp thông tin về chế độ kênh PDU hiện tại thông qua dịch vụ CanIf\_GetPduMode().



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

Hình 7.8 hiển thị một biểu đồ với các chế độ kênh PDU có thể. Mỗi kênh L-PDU có thể ở trong các chế độ CANIF\_OFFLINE (không giao tiếp), CANIF\_TX\_OFFLINE (chế độ ch passive => lắng nghe mà không gửi), CANIF\_TX\_OFFLINE\_ACTIVE (mô phỏng việc truyền với việc lắng nghe), và CANIF\_ONLINE (giao tiếp đầy đủ). Trạng thái mặc định là chế độ CANIF\_OFFLINE.

###### CANIF\_OFFLINE

[SWS\_CANIF\_00864] [Trong quá trình khởi tạo, CanIf sẽ chuyển mỗi kênh sang chế độ CANIF\_OFFLINE.]()

[SWS\_CANIF\_00865] [Nếu CanIf\_SetControllerMode(ControllerId, CAN\_- CS\_SLEEP) được gọi, CanIf sẽ đặt chế độ kênh PDU của kênh tương ứng thành CANIF\_OFFLINE.]()

[SWS\_CANIF\_00073] [Đối với Các kênh vật lý chuyển sang chế độ CANIF\_OFFLINE, CanIf sẽ:

• Ngăn chặn việc chuyển tiếp các yêu cầu truyền CanIf\_Transmit() của các L-PDU liên quan đến CanDrv (trả về E\_NOT\_OK cho các mô-đun lớp trên gọi),

• Xóa các bộ đệm truyền tương ứng của CanIf,

• Ngăn chặn việc triệu gọi dịch vụ gọi lại biểu thị nhận của các mô-đun lớp trên,

• Ngăn chặn việc triệu gọi dịch vụ gọi lại xác nhận truyền của các mô-đun lớp trên.

]()

[SWS\_CANIF\_00866] [Nếu CanIf\_SetControllerMode(ControllerId, CAN\_- CS\_STOPPED) hoặc CanIf\_ControllerBusOff(ControllerId) được gọi, CanIf sẽ đặt chế độ kênh PDU của kênh tương ứng thành CANIF\_TX\_OFF- LINE.]()

[SWS\_CANIF\_00489] [Đối với Các kênh vật lý chuyển sang chế độ CANIF\_TX\_OFFLINE, CanIf sẽ:

• Ngăn chặn việc chuyển tiếp các yêu cầu truyền CanIf\_Transmit() của các L-PDU liên quan đến CanDrv (trả về E\_NOT\_OK cho các mô-đun lớp trên gọi),

• Xóa các bộ đệm truyền tương ứng của CanIf,

• Ngăn chặn việc triệu gọi dịch vụ gọi lại xác nhận truyền của các mô-đun lớp trên.

• Cho phép việc triệu gọi dịch vụ gọi lại biểu thị nhận của các mô-đun lớp trên.

]()

Thông báo BusOff được ẩn ngầm trong trường hợp CANIF\_OFFLINE và CANIF\_TX\_OFFLINE do thực tế là không có L-PDU nào có thể được truyền và do đó Bộ điều khiển CAN không thể vào chế độ BusOff bởi các L-PDU mới được yêu cầu để truyền.

[SWS\_CANIF\_00118] [Nếu những L-PDU Truyền, đã đợi để truyền trong Đối tượng Phần cứng Truyền CAN, sẽ được truyền ngay sau khi thay đổi thành chế độ CANIF\_TX\_OFFLINE hoặc CANIF\_OFFLINE và một sự kiện BusOff sau đó xảy ra, CanIf không cấm việc thực hiện thông báo BusOff <User\_ControllerBusOff>(ControllerId). ]()

Thông báo Wake-up không bị ảnh hưởng đối với các thay đổi chế độ kênh PDU.

###### CANIF\_ONLINE

[SWS\_CANIF\_00075] [Đối với Các kênh vật lý chuyển sang chế độ CANIF\_ONLINE, CanIf sẽ:

• Kích hoạt việc chuyển tiếp các yêu cầu truyền CanIf\_Transmit() của các L-PDU liên quan đến CanDrv,

• Kích hoạt việc triệu gọi dịch vụ gọi lại biểu thị nhận của các mô-đun lớp trên,

• Kích hoạt việc triệu gọi dịch vụ gọi lại xác nhận truyền của các mô-đun lớp trên.

###### CANIF\_OFFLINE\_ACTIVE

[SWS\_CANIF\_00072] [Đối với mỗi L-PDU được gán cho một kênh nằm trong chế độ CANIF\_TX\_OFFLINE\_ACTIVE, CanIf sẽ gọi ngay lập tức các dịch vụ gọi lại xác nhận truyền của các mô-đun lớp trên thay vì đưa vào bộ nhớ đệm hoặc chuyển tiếp các L-PDU đến CanDrv trong quá trình gọi CanIf\_Transmit().]()

Ghi chú: Trong chế độ CANIF\_TX\_OFFLINE\_ACTIVE, lớp trên phải xử lý việc thực thi các xác nhận truyền ngay lập tức sau khi kết thúc yêu cầu truyền (xem [SWS\_CANIF\_00072]).

Lý do: Chức năng này hữu ích để thực hiện các chế độ hoạt động đặc biệt (ví dụ: chế độ chẩn đoán thụ động) để tránh lưu lượng truyền trên bus mà không ảnh hưởng đến cơ chế thông báo. Chế độ này thường được sử dụng cho mục đích chẩn đoán.

#### Software receive filter

Không phải tất cả các L-PDU, có thể vượt qua bộ lọc chấp nhận phần cứng và do đó được nhận thành công trong các đối tượng phần cứng BasicCAN, được xác định là L-PDU nhận và do đó không cần thiết từ ECU tương ứng. CanIf có thể lựa chọn lọc ra những L-PDU này và ngăn chặn việc xử lý phần mềm tiếp theo.

Để tối ưu hóa thời gian chạy của các bộ lọc phần mềm, một số thuật toán bộ lọc phần mềm được cung cấp. Cách tiếp cận của các cơ chế bộ lọc phần mềm là tìm ra L-PDU tương ứng từ HRH và CanId đang được xử lý. Sau khi L-PDU được tìm thấy, CanIf chấp nhận việc nhận và cho phép các tầng cao truy cập trực tiếp vào thông tin L-SDU.

##### Software filtering concept

Công cụ cấu hình xử lý thông tin về các thiết lập bộ lọc chấp nhận phần cứng. Các thiết lập quan trọng nhất là số lượng các đối tượng phần cứng L-PDU và phạm vi của chúng. Phạm vi của bộ lọc xác định, các L-PDU nhận nằm trong mỗi Đối tượng Nhận Phần Cứng. Các định nghĩa sau đây là có thể:

- Một L-PDU nhận đơn (nhận FullCAN),

- Một danh sách các L-PDU nhận hoặc

- Một hoặc nhiều phạm vi của các L-PDU nhận có thể được liên kết với một Đối tượng Nhận Phần Cứng (nhận BasicCAN).

Để định nghĩa phạm vi nhận, cần thiết phải xác định ít nhất một Rx L-PDU trong đó CanId hoặc toàn bộ phạm vi ID nằm bên trong phạm vi đã xác định.

Một phạm vi CanIds mà cần phải thông qua bộ lọc nhận phần mềm có thể được xác định bằng giới hạn trên của nó (xem CanIfHrhRangeRxPduUpperCanId) và giới hạn dưới (xem CanIfHrhRangeRxPduLowerCanId) CanId hoặc bằng một ID cơ sở (xem CanIfHrhRangeBaseId) và một mặt nạ xác định các bit liên quan của ID cơ sở (xem CanIfHrhRangeMask).

Lưu ý: Bộ lọc nhận phần mềm là tùy chọn (xem multiplicity của 0..∗ trong CanIfHrhRangeCfg).

*Các L-PDU nhận được cung cấp dưới dạng cấu trúc hằng số được tạo ra tĩnh từ ma trận truyền thông. Chúng được sắp xếp theo bộ lọc chấp nhận phần cứng tương ứng, sao cho có một danh sách đơn CanIds nhận cho mỗi Đối tượng Nhận Phần Cứng (HRH). Danh sách tương ứng có thể được suy ra bởi HRH, nếu nhiều đối tượng BasicCAN được sử dụng. Quá trình lọc tiếp theo là tìm kiếm qua một danh sách của nhiều CanIds bằng cách so sánh chúng với CanId nhận mới. Trong trường hợp phù hợp, L-PDU nhận được tạo ra từ CanId được tìm thấy.*

*Nếu CanId của L-PDU nhận được trong HRH được cấu hình để nhận, thì CanIf sẽ chấp nhận L-PDU này và thuật toán lọc phần mềm sẽ suy ra L-PDU nhận được tương ứng từ CanId được tìm thấy.*



**Figure 7.9: Software filtering example**

Nếu một phạm vi (một phần) chứa trong một phạm vi khác, hoặc một CanId đơn được chứa trong một phạm vi, bộ lọc phần mềm sẽ chọn L-PDU dựa trên các giả định sau đây:

- Một CanId đơn luôn quan trọng hơn một phạm vi.

- Một phạm vi nhỏ hơn luôn quan trọng hơn một phạm vi lớn hơn.

*]()*

##### Software filter algorithms

Việc lựa chọn các thuật toán tìm kiếm phù hợp là do việc triển khai của CanIf quyết định. Theo dải rộng các hoạt động BasicCAN nhận có sẵn từ CAN Controller, nên được khuyến nghị cung cấp một số thuật toán tìm kiếm như tìm kiếm tuyến tính, tìm kiếm bảng và/hoặc các biến thể tìm kiếm hash để cung cấp giải pháp tối ưu nhất cho hầu hết các trường hợp sử dụng.

#### Data Length Check

Giá trị Độ dài Dữ liệu nhận được được so sánh với giá trị Độ dài Dữ liệu được cấu hình của L-PDU nhận được. Giá trị Độ dài Dữ liệu được cấu hình sẽ được tạo ra từ kích thước của các byte được sử dụng trong L-PDU này. Giá trị Độ dài Dữ liệu được cấu hình có thể không nhất thiết là giá trị Độ dài Dữ liệu được xác định trong ma trận truyền thông CAN và được sử dụng bởi bên gửi L-PDU CAN này.

- \*\*SWS\_CANIF\_00026:\*\* CanIf sẽ chấp nhận tất cả các L-PDU nhận được (xem [SWS\_CANIF\_00390]) với một giá trị Độ dài Dữ liệu bằng hoặc lớn hơn giá trị Độ dài Dữ liệu được cấu hình (xem CanIfRxPduDataLength).

- \*\*SWS\_CANIF\_00902:\*\* Kiểm tra Độ dài Dữ liệu sẽ được xử lý nếu được kích hoạt toàn cầu (xem CanIfPrivateDataLengthCheck) và không bị vô hiệu hóa riêng từng PDU (xem CanIfRxPduDataLengthCheck).

Gợi ý: Nếu Kiểm tra Độ dài Dữ liệu bị vô hiệu hóa toàn cầu, thì không thể kích hoạt riêng lẻ cho mỗi PDU.

- \*\*SWS\_CANIF\_00168:\*\* Nếu Kiểm tra Độ dài Dữ liệu từ chối một L-PDU nhận được (xem [SWS\_CANIF\_00026]), CanIf sẽ báo lỗi chạy thời gian CANIF\_E\_INVALID\_DATA\_LENGTH cho dịch vụ Det\_ReportRuntimeError() của mô-đun DET.

- \*\*SWS\_CANIF\_00829:\*\* CanIf sẽ truyền giá trị Độ dài nhận được đến mô-đun lớp trên mục tiêu (xem [SWS\_CANIF\_00135]), nếu Kiểm tra Độ dài Dữ liệu được vượt qua.

- \*\*SWS\_CANIF\_00830:\*\* CanIf sẽ truyền giá trị Độ dài nhận được đến mô-đun lớp trên mục tiêu (xem [SWS\_CANIF\_00135]), nếu Kiểm tra Độ dài Dữ liệu không được cấu hình (xem CanIfPrivateDataLengthCheck và CanIfRxPduDataLengthCheck).

#### L-SDU dispatcher to upper layers

Lý do: Ở phía truyền, trình điều khiển L-SDU phải tìm ra dịch vụ gọi lại xác nhận Tx tương ứng của mô-đun lớp trên đích. Ở phía nhận, mỗi L-SDU thuộc về một mô-đun lớp trên duy nhất như là đích. Mối quan hệ này được gán tĩnh tại thời điểm cấu hình. Nhiệm vụ của trình điều khiển L-SDU bên trong CanIf là tìm ra khách hàng cho một L-SDU nhận được và gửi các chỉ báo đến lớp trên được tìm thấy. Các dịch vụ thông báo xác nhận truyền cũng như thông báo chỉ định nhận có thể tồn tại nhiều lần với các tên khác nhau được xác định trong các mô-đun lớp trên được thông báo. Những dịch vụ thông báo này được cấu hình tĩnh, tùy thuộc vào các lớp cần phục vụ.

#### Polling mode

Chế độ kiểm phiếu cung cấp xử lý các sự kiện truyền, nhận và lỗi xảy ra trong phần cứng CAN mà không sử dụng các ngắt phần cứng. Do đó, CanIf và CanDrv cung cấp các dịch vụ thông báo để phát hiện và thực thi các sự kiện phần cứng tương ứng. Trong chế độ kiểm phiếu, hành vi của các dịch vụ thông báo CanIf này không thay đổi. Như vậy, các mô-đun lớp trên được trừu tượng hóa khỏi chiến lược phát hiện sự kiện phần cứng. Nếu sử dụng các CanDrv khác nhau, tần suất gọi phải được điều chỉnh trong quá trình thiết lập cấu hình và tích hợp hệ thống.

Các dịch vụ thông báo này có khả năng phát hiện các sự kiện mới đã xảy ra trong các đối tượng phần cứng CAN kể từ lần thực thi trước đó. Các dịch vụ thông báo của CanIf để chuyển tiếp các sự kiện phát hiện bởi CanDrv giống như cho hoạt động ngắt (xem phần 8.4 “Thông báo gọi lại”).

Người dùng cần xem xét rằng CanIf phải có khả năng thực hiện các dịch vụ thông báo được kích hoạt bởi ngắt ở cấp độ ngắt cũng như thực hiện các dịch vụ thông báo được gọi tại cấp độ nhiệm vụ. Nếu truy cập vào hộp thư của bộ điều khiển CAN bị chặn, việc đệm truyền sau đó sẽ diễn ra (tham khảo phần 7.11 “Đệm truyền”).

Chế độ Kiểm phiếu và Ngắt có thể được cấu hình cho mỗi bộ điều khiển CAN cơ bản.

#### Multiple CAN Driver support

CanIf cần một ánh xạ cụ thể để bao phủ nhiều CanDrv để cung cấp một giao diện chung cho các lớp trên. Do đó, CanIf phải chuyển tiếp tất cả các hành động lên xuống các API của các CanDrv tương ứng và các CAN Controller(s) cơ bản. Đối với cách thức xuống lên, CanIf phải cung cấp các thông báo gọi lại thích hợp để phân biệt giữa nhiều CanDrvs.

Mỗi CanDrv hỗ trợ một số lượng nhất định các CAN Controllers cơ bản và một số cố định các HTHs/HRHs. Mỗi CanDrv có một khu vực đánh số riêng, luôn bắt đầu từ số không cho các CAN Controllers và HTHs. CanIf phải suy ra được CanDrv tương ứng từ L-SDU được truyền trong các API. Các tham số phải được dịch tương ứng: ví dụ như L-SDU => HTH/HRH, CanId, Độ dài dữ liệu."

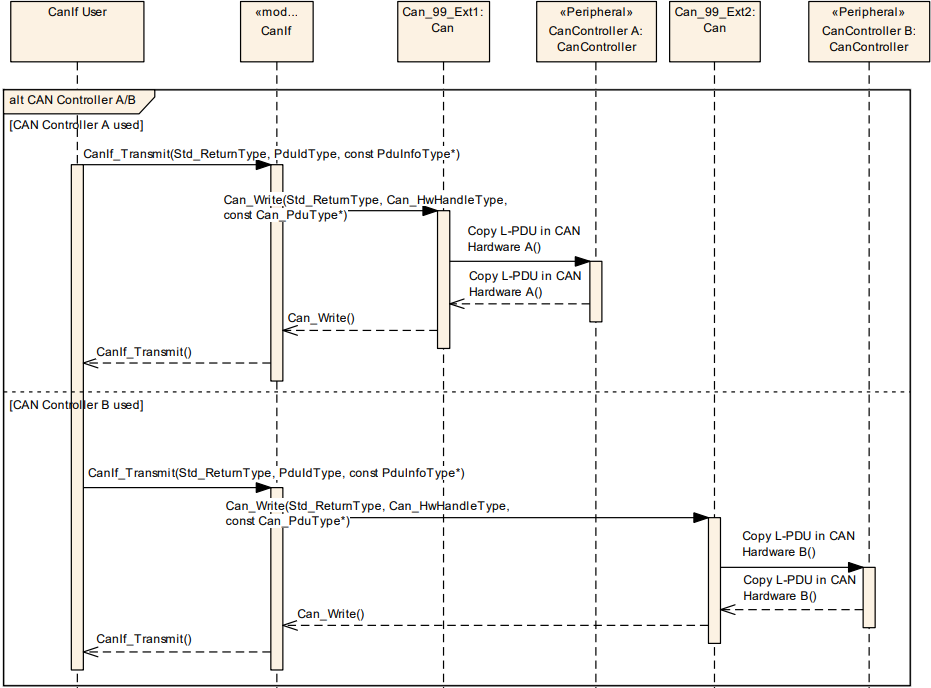
Hỗ trợ cho nhiều CanDrvs có thể được bật và tắt bằng tham số cấu hình CanIfPublicMultipleDrvSupport.

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

Mỗi L-PDU truyền gửi cho phép CanIf suy ra được CAN Controller tương ứng và mặt ngầm CanDrv đang phục vụ Đơn vị Phần cứng bị ảnh hưởng. Giải quyết các phụ thuộc này là có thể vì cấu trúc của Bộ Điều khiển CAN: nó kết hợp CanDrv Handle và CAN Controller tương ứng trong Đơn vị Phần cứng.

Tại thời điểm cấu hình, một Bộ Điều khiển CAN sẽ được ánh xạ cho mỗi CAN Controller. Biểu đồ chuỗi trong Hình 7.10 dưới đây minh họa hai yêu cầu truyền khác nhau được chỉ định cho các CanDrvs khác nhau. CanIf chỉ cần chọn CanDrv tương ứng để gọi dịch vụ API đúng.

Lưu ý: Hình 7.10 và bảng tiếp theo chỉ là một ví dụ. Cuối cùng, việc truy cập các API đúng của CanDrvs dưới mức độ triển khai.

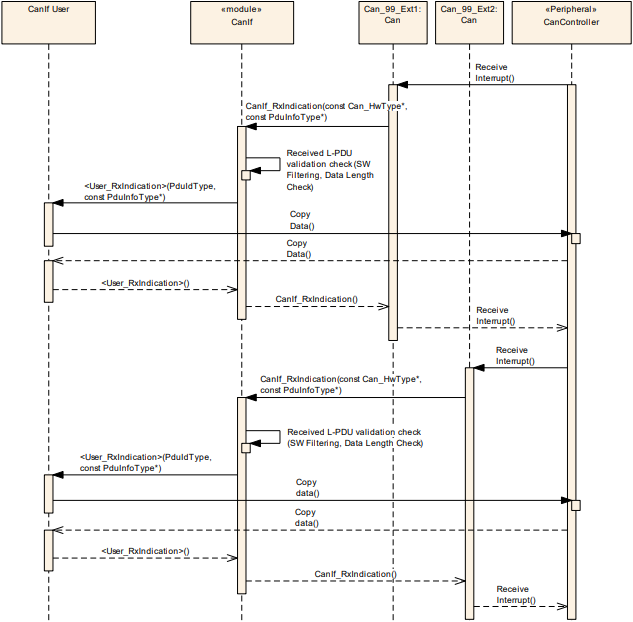


**Figure 7.10:** **Transmission request with multiple CAN Drivers - simplified**

|  |  |
| --- | --- |
| **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

Ngay cả khi nhiều CanDrv được sử dụng trong một ECU duy nhất, Mỗi dịch vụ gọi lại thông báo được kích hoạt bởi CanDrvs tại CanIf tồn tại chỉ một lần. Điều này có nghĩa là, CanIf phải xác định CanDrv gọi từ các thông số được truyền. CanIf xác định CanDrv gọi từ ControllerId trong cấu trúc Mailbox (Can\_HwType).



**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] [Nếu Mạng Nối Bộ Phận (PN) được kích hoạt (xem CanIfPublicPn- Support), CanIf sẽ hỗ trợ một PnTxFilter cho mỗi Bộ Điều Khiển CAN nằm phía trên lớp PDU.]()

[SWS\_CANIF\_00748] [PnTxFilter của [SWS\_CANIF\_00747] chỉ có hiệu lực và chuyển đổi các chế độ của nó (được bật/tắt) nếu có nhiều hơn không Tx L-PDUs trên mỗi Bộ Điều Khiển CAN được cấu hình như CanIfTxPduPnFilterPdu (xem CanIfTx- PduPnFilterPdu).]()

[SWS\_CANIF\_00863] [PnTxFilter phải được bật trong quá trình khởi tạo (tham khảo [SWS\_CANIF\_00747] và [SWS\_CANIF\_00748]).]()

[SWS\_CANIF\_00749] [Nếu CanIf\_SetControllerMode(ControllerId, CAN\_- CS\_SLEEP) được gọi thì PnTxFilter của Bộ Điều Khiển CAN tương ứng sẽ được kích hoạt (tham khảo [SWS\_CANIF\_00748] và [SWS\_CANIF\_00747]).]()

[SWS\_CANIF\_00750] [Nếu PnTxFilter của một Bộ Điều Khiển CAN được bật, CanIf sẽ chặn tất cả các yêu cầu truyền đi đến Bộ Điều Khiển CAN đó (trả về E\_NOT\_OK khi CanIf\_- Transmit() được gọi), trừ khi L-PDU truyền yêu cầu là một trong các CanIfTxPduPnFilterPdus được cấu hình của Bộ Điều Khiển CAN đó. Những CanIfTxPduPnFil- terPdus này luôn được chuyển đến Bộ Điều Khiển CAN tương ứng.]()

[SWS\_CANIF\_00751] [Nếu CanIf\_TxConfirmation() được gọi, thì PnTxFilter tương ứng sẽ được tắt (tham khảo [SWS\_CANIF\_00747] và [SWS\_CANIF\_00748]).]()

[SWS\_CANIF\_00896] [Nếu CanIf\_RxIndication() được gọi và PnTxFilter được bật, thì PnTxFilter tương ứng sẽ được tắt (tham khảo [SWS\_CANIF\_00747] và [SWS\_CANIF\_00748]).]()

[SWS\_CANIF\_00752] [Nếu PnTxFilter của một Bộ Điều Khiển CAN được tắt, CanIf sẽ hoạt động theo yêu cầu thông qua CanIf\_SetPduMode() (xem [SWS\_CANIF\_00008]).] ()

[SWS\_CANIF\_00878] [Nếu CanIf\_SetPduMode(ControllerId, CANIF\_TX\_OF- FLINE) được gọi và Mạng Nối Bộ Phận được kích hoạt (tham khảo CanIfPublicPnSup- port) thì PnTxFilter của Bộ Điều Khiển CAN tương ứng sẽ được kích hoạt (tham khảo [SWS\_CANIF\_00748] và [SWS\_CANIF\_00747]).]()

#### CAN FD Support

Vì lý do hiệu suất, một số Bộ Điều Khiển CAN cho phép sử dụng tính năng Tốc Độ Dữ Liệu Linh Hoạt gọi là CAN FD (xem [12, ISO 11898-1:2015]). Ngoài ra, với tốc độ baud cao hơn cho dữ liệu, CAN FD cũng hỗ trợ một bộ dữ liệu mở rộng cho phép truyền lên đến 64 byte. Việc các tính năng này có sẵn hay không phụ thuộc vào sự hỗ trợ CAN FD chung của Bộ Điều Khiển CAN và nếu Bộ Điều Khiển CAN đang ở chế độ CAN FD (Cấu hình Tốc Độ Baudrate CanControllerFdBaudrateConfig hợp lệ).

Nếu một L-SDU được gửi dưới dạng khung CAN FD hoặc khung CAN 2.0 truyền thống phụ thuộc vào loại CanIfTxPduCanIdType đã được cấu hình. CanIf thông báo điều này cho CanDrv bằng cách sử dụng bit thứ hai quan trọng nhất của PduInfo->id (Can\_IdType) được truyền trong quá trình gọi Can\_Write().

Note: Nếu CanDrv không ở chế độ CAN FD (không có CanControllerFdBaudrateConfig, thì L-PDU sẽ được gửi dưới dạng khung CAN 2.0 truyền thống miễn là SduLength <= 8 byte.

Note: Pha cạnh tranh của các khung CAN 2.0 truyền thống và các khung CAN FD không khác biệt nếu sử dụng cùng một CanId. Do đó, ngay cả khi sử dụng các khung CAN FD, mỗi CanId không được sử dụng nhiều hơn một lần.

Loại khung nào đã được nhận bởi CanDrv cũng được chỉ ra bằng cách sử dụng bit thứ hai quan trọng nhất của Can\_IdType được truyền với CanIf\_RxIndication() (Mailbox-

->CanId). Dựa trên thông tin này, CanIf quyết định cách ánh xạ vào các L-SDU đã được cấu hình (CanIfRxPduCfg) như mô tả trong [SWS\_CANIF\_00877].

Note: Nếu các tầng trên không quan tâm liệu một tin nhắn đã được nhận bằng khung CAN 2.0 truyền thống hay khung CAN FD, có thể chỉ cần sử dụng một CanIfRxPduCfg cho cả hai loại (xem CanIfRxPduCanIdType). Điều này có thể cho phép tối ưu hóa cục bộ. Tuy nhiên, từ góc nhìn của hệ thống, định dạng cho mỗi khung phải được cấu hình. Nếu không, người gửi sẽ không biết loại khung nào sẽ được truyền.

#### 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 |

[SWS\_CANIF\_00913] [Nếu việc báo cáo sự kiện bảo mật đã được kích hoạt cho mô-đun CanIf (CanIfEnableSecurityEventReporting = true), các sự kiện bảo mật tương ứng sẽ được báo cáo cho IdsM qua các giao diện được xác định trong AUTOSAR\_SWS- BSWGeneral.](RS\_Ids\_00810)

[SWS\_CANIF\_00915] [Nếu CanIf\_ErrorNotification() được gọi bởi CanDrv, hàm sẽ đánh giá xem có phát hiện lỗi liên quan đến việc truyền không. Trong trường hợp này, CanIf sẽ báo cáo sự kiện bảo mật CANIF\_SEV\_TX\_ERROR\_DETECTED. Dữ liệu ngữ cảnh được cấu trúc như sau:

Dữ liệu ngữ cảnh (2 byte)

• ControllerID (1 byte)

• CanError (1 byte)

– CAN\_ERROR\_BIT\_MONITORING1 (0x1)

– CAN\_ERROR\_BIT\_MONITORING0 (0x2)

– CAN\_ERROR\_BIT (0x3)

– CAN\_ERROR\_CHECK\_ACK\_FAILED (0x4)

– CAN\_ERROR\_ACK\_DELIMITER (0x5)

– CAN\_ERROR\_ARBITRATION\_LOST (0x6)

– CAN\_ERROR\_OVERLOAD (0x7)](RS\_Ids\_00810)

*[SWS\_CANIF\_00913] [Nếu việc báo cáo sự kiện bảo mật đã được kích hoạt cho mô-đun CanIf (CanIfEnableSecurityEventReporting = true), các sự kiện bảo mật tương ứng sẽ được báo cáo cho IdsM qua các giao diện được xác định trong AUTOSAR\_SWS- BSWGeneral.](RS\_Ids\_00810)*

*[SWS\_CANIF\_00915] [Nếu CanIf\_ErrorNotification() được gọi bởi CanDrv, hàm sẽ đánh giá xem có phát hiện lỗi liên quan đến việc truyền không. Trong trường hợp này, CanIf sẽ báo cáo sự kiện bảo mật CANIF\_SEV\_TX\_ERROR\_DETECTED. Dữ liệu ngữ cảnh được cấu trúc như sau:*

*Dữ liệu ngữ cảnh (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)*

#### 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*** | Loại này xác định một cấu trúc dữ liệu cho các thông số xây dựng sau của CAN Interface cho tất cả các trình điều khiển CAN cơ bản. Trong quá trình khởi tạo, CanIf nhận một con trỏ đến một cấu trúc của loại này để truy cập dữ liệu cấu hình của nó, điều này là cần thiết cho quá trình khởi tạo. | |
| ***Available via*** | CanIf.h | |

[**SWS\_CANIF\_00523]** [Cấu trúc dữ liệu khởi tạo cho một loại cụ thể CanIf\_ConfigType sẽ bao gồm định nghĩa của các thông số công khai CanIf và định nghĩa cho mỗi L-PDU/L-SDU.]()

Chú ý: Định nghĩa của các thông số công khai CanIf và định nghĩa cho mỗi L-PDU/L-SDU được chỉ định trong chương 10.

##### CanIf\_PduModeType

**[SWS\_CANIF\_00137]**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name*** | CanIf\_PduModeType | | |
| ***Kind*** | Enumeration | | |
| ***Range*** | CANIF\_OFFLINE | 0x00 | = 0 Đường truyền truyền và nhận tương ứng của kênh đó bị vô hiệu hóa => chế độ không truyền thông |
| CANIF\_TX\_OFFLINE | 0x01 | Đường truyền tương ứng của kênh đó bị vô hiệu hóa. Đường nhận được kích hoạt. |
| CANIF\_TX\_OFFLINE\_ ACTIVE | 0x02 | Đường truyền tương ứng của kênh đó đang ở chế độ hoạt động ngoại tuyến tích cực (xem SWS\_CANIF\_00072). Đường nhận được kích hoạt. Chế độ này yêu cầu CanIfTxOfflineActive Support = TRUE. |
| CANIF\_ONLINE | 0x03 | Đường truyền và đường nhận tương ứng của kênh đó được kích hoạt => chế độ hoạt động đầy đủ |
| ***Description*** | Chế độ PDU của một kênh xác định hoạt động truyền hoặc nhận của nó. Hướng truyền thông (truyền và/hoặc nhận) của kênh có thể được điều khiển riêng biệt hoặc cùng nhau bởi các lớp trên cùng. | | |
| ***Available via*** | CanIf.h | | |

##### CanIf\_NotifStatusType

**[SWS\_CANIF\_00201]**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name*** | CanIf\_NotifStatusType | | |
| ***Kind*** | Enumeration | | |
| ***Range*** | CANIF\_TX\_RX\_ NOTIFICATION | – | CAN L-PDU được yêu cầu đã được truyền hoặc nhận thành công. |
| CANIF\_NO\_NOTIFICATION | 0x00 | Không có sự kiện truyền hoặc nhận nào xảy ra đối với L-PDU được yêu cầu. |
| ***Description*** | Giá trị trả về của trạng thái thông báo CAN L-PDU. | | |
| ***Available via*** | CanIf.h | | |

#### Function definitions

[**SWS\_CANIF\_00661**] [Tất cả các dịch vụ API của CanIf, ngoại trừ CanIf\_Init() và CanIf\_GetVersionInfo(), sẽ không thực hiện hoạt động bình thường và trả về E\_NOT\_OK trừ khi CanIf đã được khởi tạo bằng một cuộc gọi trước đó của CanIf\_Init().

##### 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

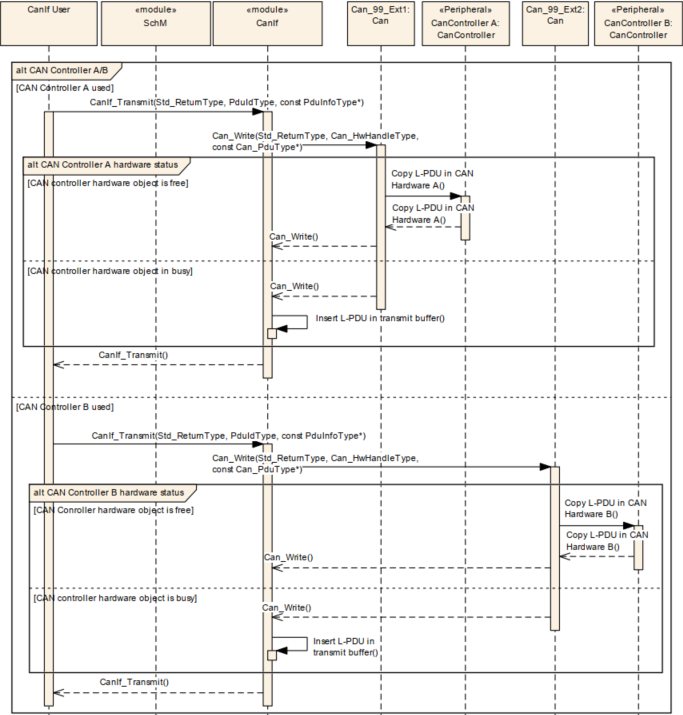
CanIf User

«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**

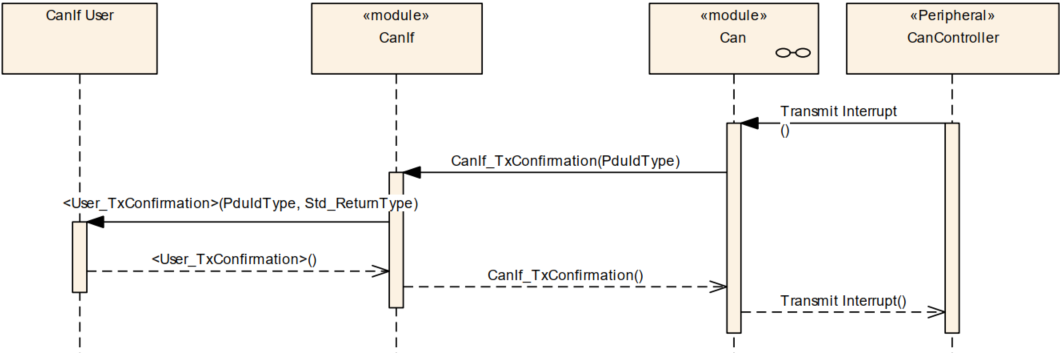
First transmit request:

|  |  |
| --- | --- |
| **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. |

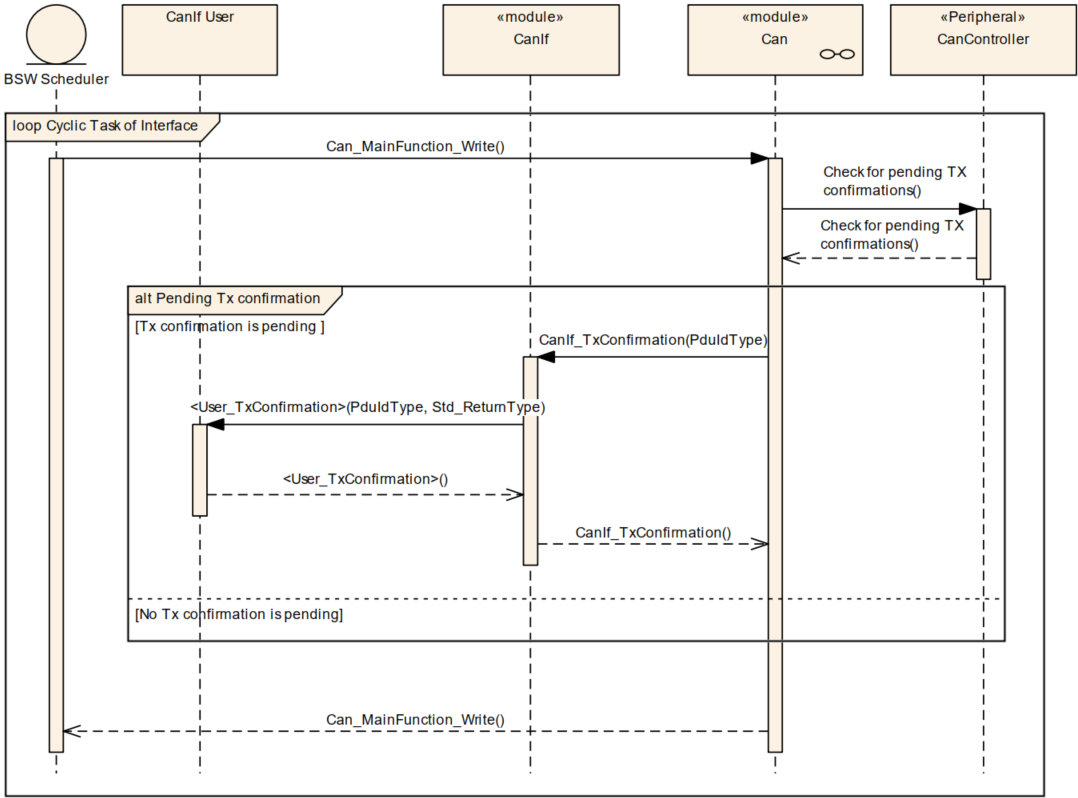
#### Transmit confirmation (interrupt mode)

****

**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) |  |  |  |

)