<table>
<thead>
<tr>
<th>Document Title</th>
<th>Specification of CAN Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Owner</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>Document Responsibility</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>Document Identification No</td>
<td>011</td>
</tr>
<tr>
<td>Document Status</td>
<td>Final</td>
</tr>
<tr>
<td>Part of AUTOSAR Standard</td>
<td>Classic Platform</td>
</tr>
<tr>
<td>Part of Standard Release</td>
<td>4.3.1</td>
</tr>
</tbody>
</table>

**Document Change History**

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-12-08</td>
<td>4.3.1</td>
<td>AUTOSAR Release Management</td>
<td>- Added Support to Tx/RxProcessing per Controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Incompatible return types are corrected to E_NOT_OK and E_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Can_StateTransitionType is removed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Runtime error is added and Rephrased from &quot;default error&quot; to &quot;development error&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- SWS_CAN_00504 and SWS_Can_00416 is modified</td>
</tr>
</tbody>
</table>
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 2016-11-30 | 4.3.0   | AUTOSAR Release Management  | - Added API’s Can_GetControllerErrorState, Can_DeInit, Can_GetControllerMode, Types Can_ControllerStateType, Can_ErrorStateType and new requirements Can_91002 to SWS_Can_91018.  
- Modified minimum range of MainFunctionPeriod parameters and replaced Word “DLC” by “Data Length”.  
- Removed unresolved BSW SRS references, definition of the “configuration variants”, Can_StateTransitionType, WAKEUP related, Can_ChangeBaudrate API support, MISRA references, requirements related to module initialization check for scheduled functions.  
- Small improvements and minor bug fixes. |
| 2015-07-31 | 4.2.2   | AUTOSAR Release Management  | - CanHwObjectCount parameter multiplicity is changed to 1  
- Error Classification has changed  
- Improved 8.4.2 Enabling/Disabling wakeup notification  
- DET has been renamed from "Development Error Tracer" to "Default Error Tracer  
- Small improvements and minor bug fixes |
<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-10-31</td>
<td>4.2.1</td>
<td>AUTOSAR Release Management</td>
<td>• Full CAN FD Support (incl. Trigger Transmit)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed CanIf_CancelTxConfirmation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Time-out and wake up event handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Small improvements and minor bug-fixes</td>
</tr>
<tr>
<td>2014-03-31</td>
<td>4.1.3</td>
<td>AUTOSAR Release Management</td>
<td>• Added new requirements SWS_CAN_00497, SWS_CAN_00498, SWS_CAN_00499, and SWS_CAN_00496</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Modified requirements ECUC_Can_00445, SWS_CAN_00487, SWS_CAN_00469, SWS_CAN_00475, and SWS_CAN_00479</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed requirements SWS_CAN_00476, and SWS_Can_00414</td>
</tr>
<tr>
<td>2013-10-31</td>
<td>4.1.2</td>
<td>AUTOSAR Release Management</td>
<td>• Removed the 'Timing' row from the API table(s) of chapter 'Scheduled Functions'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Modified range of Can_IdType and CAN_CHANGE_BAUDRATE_SUPPORT to CAN_CHANGE_BAUDRATE_API</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Editorial changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed chapter(s) on change documentation</td>
</tr>
</tbody>
</table>
### Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 2013-03-15 | 4.1.1   | AUTOSAR Administration    | • Added support for Pretended Networking  
• Add DET error CAN_E_PARAM_BAUDRATE to the error classification table  
• Corrected the sequence for EcuM_SetWakeuEvent in section 7.7  
• Updated Can_CheckWakeup as Configurable API  
• Added support to have more than one CanMailbox per HRH in order to receive back to back messages  
• Can_ChangeBaudrate and Can_CheckBaudrate API are deprecated and will be replaced by Can_SetBaudrate API |
### Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011-12-22</td>
<td>4.0.3</td>
<td>AUTOSAR Administration</td>
<td>• Added SWS_Can_00461 to capture Detection of Power ON of controller due to CAN communication</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Changed Can_InitController to Can_ChangeBaudrate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added Can_CheckBaudrate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added sub container CanMainFunctionRWPeriods to CanGeneral</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Changed CanHardwareObject container</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated description of ECUC_Can_00321</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Changed Can_SetControllerMode in SWS_Can_00370 to Can_Mainfunction_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added CanControllerDefaultBaudrate parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated description of SWS_Can_00279</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated description of CAN321</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added SWS_Can_00445, SWS_Can_00446 and SWS_Can_00447 to capture Possible loss of CAN Wakeup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Changed “Module Short Name” (MODULENAME) to “Module Abbreviation” (MAB)</td>
</tr>
</tbody>
</table>
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-12-18</td>
<td>4.0.1</td>
<td>AUTOSAR Administration</td>
<td>• Modified SWS_Can_00111 to correct the “Version Checking” information&lt;br&gt;• Added new requirements SWS_Can_00435 to SWS_Can_00440 to introduce Can_GeneralTypes.h.&lt;br&gt;• Added new requirements SWS_Can_00441 and SWS_Can_00442 to introduce multiple poll cycles&lt;br&gt;• Added new requirements SWS_Can_00443 and SWS_Can_00444 to provide an optional callback on every reception of a LPDU</td>
</tr>
<tr>
<td>2010-02-02</td>
<td>3.1.4</td>
<td>AUTOSAR Administration</td>
<td>• General improvements of requirements in preparation of CT-development.&lt;br&gt;• Can_MainFunction_Mode added to support asynchronous controller state change&lt;br&gt;• Limited number of supported message objects removed&lt;br&gt;• Description of CAN controller state transitions improved&lt;br&gt;• Debugging concept added&lt;br&gt;• Legal disclaimer revised</td>
</tr>
<tr>
<td>2008-08-13</td>
<td>3.1.1</td>
<td>AUTOSAR Administration</td>
<td>• Legal disclaimer revised</td>
</tr>
<tr>
<td>2008-02-01</td>
<td>3.0.2</td>
<td>AUTOSAR Administration</td>
<td>• Table formatting corrected</td>
</tr>
</tbody>
</table>
### Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
</table>
| 2007-12-21 | 3.0.1   | AUTOSAR Administration | • Tables generated from UML-models,  
• General improvements of requirements in preparation of CT-development.  
• Functions Can_MainFunction_Write, Can_MainFunction_Read, Can_MainFunction_BusOff and Can_MainFunction_WakeUp changed to scheduled functions  
• Cycle Parameters added for new scheduled functions  
• Wakeup concept added (Chapter REF _Ref395085489 \r \h ) and addition of function Can_Cbk_CheckWakeup  
• Document meta information extended  
• Small layout adaptations made |
### Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-01-24</td>
<td>2.1.15</td>
<td>AUTOSAR Administration</td>
<td>- File structure reworked (chapter REF_Ref158085666 \r \h )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Removed return value CAN_WAKEUP in function Can_SetControllerMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Replaced by CAN_NOT_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Renamed CanIf_ControllerWakeup to CanIf_SetWakeupEvent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Reworked development errors (chapter REF_Ref182101189 \r \h )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Removed implementation specific description in Can_Write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Changed timing of cyclic functions to “fixed cyclic”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Reworked “Scope” for all configuration variables (chapter REF_Ref104709655 \r \h )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Legal disclaimer revised</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Release notes added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- “Advice for users” revised</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- “Revision Information” added</td>
</tr>
<tr>
<td>2006-05-16</td>
<td>2.0</td>
<td>AUTOSAR Administration</td>
<td>- Document structure adapted to common Release 2.0 SWS Template</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- clarified development and production error handling and function abortion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- multiplexed transmission and TX cancellation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- version check</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- configuration description according template</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- individual main functions for RX TX and status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
## Document Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Changed by</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005-05-31</td>
<td>1.0</td>
<td>AUTOSAR Administration</td>
<td>• Initial release</td>
</tr>
</tbody>
</table>
Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.
# Table of Content

1. Introduction and functional overview ................................................................. 14
2. Acronyms and abbreviations .................................................................................. 15
   2.1 Priority Inversion ............................................................................................... 16
   2.2 CAN Hardware Unit .......................................................................................... 18
3. Related documentation ......................................................................................... 19
   3.1 Input documents ............................................................................................... 19
   3.2 Related standards and norms ........................................................................... 20
   3.3 Related specification ......................................................................................... 20
4. Constraints and assumptions ................................................................................ 21
   4.1 Limitations ........................................................................................................ 21
   4.2 Applicability to car domains ............................................................................ 21
5. Dependencies to other modules .......................................................................... 22
   5.1 Static Configuration .......................................................................................... 22
   5.2 Driver Services ................................................................................................. 22
   5.3 System Services ............................................................................................... 22
   5.4 Can module Users ............................................................................................ 23
   5.5 File structure .................................................................................................... 24
      5.5.1 Header file structure ................................................................................... 24
6. Requirements traceability ..................................................................................... 26
7. Functional specification ......................................................................................... 33
   7.1 Driver scope ..................................................................................................... 33
   7.2 Driver State Machine ...................................................................................... 34
   7.3 CAN Controller State Machine ........................................................................ 34
      7.3.1 CAN Controller State Description ............................................................. 35
      7.3.2 CAN Controller State Transitions .............................................................. 36
      7.3.3 State transition caused by function Can_Init ........................................... 37
      7.3.4 State transition caused by function Can_SetBaudrate ............................ 38
      7.3.5 State transition caused by function Can_SetControllerMode ......... 38
      7.3.6 State transition caused by Hardware Events ........................................... 40
      7.3.7 State transition caused by function Can_DelInit .................................. 41
   7.4 Can module/Controller Initialization ............................................................... 41
   7.5 L-PDU transmission ......................................................................................... 42
      7.5.1 Priority Inversion ...................................................................................... 43
      7.5.2 Transmit Data Consistency ....................................................................... 44
   7.6 L-PDU reception ............................................................................................... 45
      7.6.1 Receive Data Consistency ......................................................................... 45
   7.7 Wakeup concept ............................................................................................... 47
   7.8 Notification concept ....................................................................................... 47
   7.9 Reentrancy issues ............................................................................................ 48
   7.10 Pretended Networking ................................................................................... 48
      7.10.1 Support Pretended Networking mode handling .................................... 49
      7.10.2 Support autonomous sending and receiving of messages ............. 50
7.11 Error classification
7.11.1 Development Errors
7.11.2 Runtime Errors
7.11.3 Runtime Errors
7.11.4 Transient Faults
7.11.5 Production Errors
7.11.6 Return Values
7.12 CAN FD Support

8 API specification
8.1 Imported types
8.2 Type definitions
8.2.1 Can_ConfigType
8.2.2 Can_PduType
8.2.3 Can_IdType
8.2.4 Can_HwHandleType
8.2.5 Can_HwType
8.2.6 Can_ReturnType
8.2.7 Can_ErrorStateType
8.2.8 Can_ControllerStateType
8.3 Function definitions
8.3.1 Services affecting the complete hardware unit
8.3.1.1 Can_Init
8.3.1.2 Can_GetVersionInfo
8.3.1.3 Can_DeInit
8.3.2 Services affecting one single CAN Controller
8.3.2.1 Can_SetBaudrate
8.3.2.2 Can_SetControllerMode
8.3.2.3 Can_DisableControllerInterrupts
8.3.2.4 Can_EnableControllerInterrupts
8.3.2.5 Can_CheckWakeup
8.3.2.6 Can_GetControllerErrorState
8.3.2.7 Can_GetControllerMode
8.3.3 Services affecting a Hardware Handle
8.3.3.1 Can_Write
8.4 Call-back notifications
8.4.1 Call-out function
8.4.2 Enabling/Disabling wakeup notification
8.5 Scheduled functions
8.5.1.1 Can_MainFunction_Write
8.5.1.2 Can_MainFunction_Read
8.5.1.3 Can_MainFunction_BusOff
8.5.1.4 Can_MainFunction_Wakeup
8.5.1.5 Can_MainFunction_Mode
8.6 Expected Interfaces
8.6.1 Mandatory Interfaces
8.6.2 Optional Interfaces
8.6.3 Configurable interfaces
8.7 API supporting Pretended Networking

- AUTOSAR confidential -
9  Sequence diagrams ................................................................. 74
9.1  Interaction between Can and CanIf module ........................... 74
9.2  Wakeup sequence ............................................................... 74

10  Configuration specification .................................................. 75
10.1  How to read this chapter ..................................................... 75
10.2  Containers and configuration parameters ............................ 75
  10.2.1  Can ................................................................. 85
  10.2.2  CanGeneral .......................................................... 85
  10.2.3  CanController ......................................................... 89
  10.2.4  CanControllerBaudrateConfig .................................... 93
  10.2.5  CanControllerFdBaudrateConfig .................................. 95
  10.2.6  CanHardwareObject ................................................ 97
  10.2.7  CanHwFilter .......................................................... 101
  10.2.8  CanConfigSet ......................................................... 102
  10.2.9  CanMainFunctionRWPeriods ....................................... 102
  10.2.10 CanIcom .............................................................. 103
  10.2.11 CanIcomConfig ....................................................... 103
  10.2.12 CanIcomGeneral ..................................................... 104
  10.2.13 CanIcomRxMessage ................................................ 105
  10.2.14 CanIcomRxMessageSignalConfig .................................. 107
  10.2.15 CanIcomWakeupCauses ............................................ 109

11  Not applicable requirements ................................................ 110
1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module CAN Driver (called “Can module” in this document).

The Can module is part of the lowest layer, performs the hardware access and offers a hardware independent API to the upper layer.

The only upper layer that has access to the Can module is the CanIf module (see also SRS_SPAL_12092).

The Can module provides services for initiating transmissions and calls the callback functions of the CanIf module for notifying events, independently from the hardware.

Furthermore, it provides services to control the behavior and state of the CAN controllers that are belonging to the same CAN Hardware Unit.

Several CAN controllers can be controlled by a single Can module as long as they belong to the same CAN Hardware Unit.

For a closer description of CAN controller and CAN Hardware Unit see chapter Acronyms and abbreviations and a diagram in [5].
# 2 Acronyms and abbreviations

<table>
<thead>
<tr>
<th>Abbreviation / Acronym:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN controller</td>
<td>A CAN controller serves exactly one physical channel.</td>
</tr>
<tr>
<td>CAN Hardware Unit</td>
<td>A CAN Hardware Unit may consist of one or multiple CAN controllers of the same type and one or multiple CAN RAM areas. The CAN Hardware Unit is either on-chip, or an external device. The CAN Hardware Unit is represented by one CAN driver.</td>
</tr>
<tr>
<td>CAN L-PDU</td>
<td>Data Link Layer Protocol Data Unit. Consists of Identifier, Data Length and Data (SDU). (see[18])</td>
</tr>
<tr>
<td>CAN L-SDU</td>
<td>Data Link Layer Service Data Unit. Data that is transported inside the L-PDU. (see[18])</td>
</tr>
<tr>
<td>DLC</td>
<td>Data Length Code (part of CAN message describes the SDU length)</td>
</tr>
<tr>
<td>Hardware Object</td>
<td>A CAN hardware object is defined as a PDU buffer inside the CAN RAM of the CAN hardware unit / CAN controller. A Hardware Object is defined as L-PDU buffer inside the CAN RAM of the CAN Hardware Unit.</td>
</tr>
<tr>
<td>Hardware Receive Handle (HRH)</td>
<td>The Hardware Receive Handle (HRH) is defined and provided by the CAN Driver. Each HRH typically represents just one hardware object. The HRH can be used to optimize software filtering.</td>
</tr>
<tr>
<td>Hardware Transmit Handle (HTH)</td>
<td>The Hardware Transmit Handle (HTH) is defined and provided by the CAN Driver. Each HTH typically represents just one or multiple hardware objects that are configured as hardware transmit buffer pool.</td>
</tr>
<tr>
<td>Inner Priority Inversion</td>
<td>Transmission of a high-priority L-PDU is prevented by the presence of a pending low-priority L-PDU in the same transmit hardware object.</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td>L-PDU Handle</td>
<td>The L-PDU handle is defined and placed inside the CanIf module layer. Typically each handle represents an L-PDU, which is a constant structure with information for Tx/Rx processing.</td>
</tr>
<tr>
<td>MCAL</td>
<td>Microcontroller Abstraction Layer</td>
</tr>
<tr>
<td>Outer Priority Inversion</td>
<td>A time gap occurs between two consecutive transmit L-PDUs. In this case a lower priority L-PDU from another node can prevent sending the own higher priority L-PDU. Here the higher priority L-PDU cannot participate in arbitration during network access because the lower priority L-PDU already won the arbitration.</td>
</tr>
<tr>
<td>Physical Channel</td>
<td>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.</td>
</tr>
<tr>
<td>Priority</td>
<td>The Priority of a CAN L-PDU is represented by the CAN Identifier. The lower the numerical value of the identifier, the higher the priority.</td>
</tr>
<tr>
<td>SFR</td>
<td>Special Function Register. Hardware register that controls the controller behavior.</td>
</tr>
<tr>
<td>SPAL</td>
<td>Standard Peripheral Abstraction Layer</td>
</tr>
<tr>
<td>ICOM</td>
<td>Intelligent Communication Controller</td>
</tr>
</tbody>
</table>
2.1 Priority Inversion

“If only a single transmit buffer is used inner priority inversion may occur. Because of low priority a message stored in the buffer waits until the “traffic on the bus calms down”. During the waiting time this message could prevent a message of higher priority generated by the same microcontroller from being transmitted over the bus.”

\[1\] Picture and text by CiA (CAN in Automation)
“The problem of outer priority inversion may occur in some CAN implementations. Let us assume that a CAN node wishes to transmit a package of consecutive messages with high priority, which are stored in different message buffers. If the interframe space between these messages on the CAN network is longer than the minimum space defined by the CAN standard, a second node is able to start the transmission of a lower priority message. The minimum interframe space is determined by the Intermission field, which consists of 3 recessive bits. A message, pending during the transmission of another message, is started during the Bus Idle period, at the earliest in the bit following the Intermission field. The exception is that a node with a waiting transmission message will interpret a dominant bit at the third bit of Intermission as Start-of-Frame bit and starts transmission with the first identifier bit without first transmitting an SOF bit. The internal processing time of a CAN module has to be short enough to send out consecutive messages with the minimum interframe space to avoid the outer priority inversion under all the scenarios mentioned.”²

² Text and image by CiA (CAN in Automation)
2.2 CAN Hardware Unit

The CAN Hardware Unit combines one or several CAN controllers, which may be located on-chip or as external standalone devices of the same type, with common or separate Hardware Objects. Following figure shows a CAN Hardware Unit consisting of two CAN controllers connected to two Physical Channels:
3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
   AUTOSAR_EXP_LayeredSoftwareArchitecture..pdf

   AUTOSAR_SRS_BSWGeneral.pdf

   AUTOSAR_SRS_SPALGeneral.pdf

[4] Requirements on CAN
   AUTOSAR_SRS_CAN.pdf

   AUTOSAR_SWS_CANInterface.pdf

   AUTOSAR_SWS_DefaultErrorTracer.pdf

[7] Specification of ECU State Manager
   AUTOSAR_SWS_ECUStateManager.pdf

[8] Specification of MCU Driver
   AUTOSAR_SWS_MCUDriver.pdf

   AUTOSAR_SWS_OS.pdf

    AUTOSAR_TPS_ECUConfiguration.pdf

    AUTOSAR_SWS_SPIHandlerDriver.doc.pdf

[12] Specification of Memory Mapping
    AUTOSAR_SWS_MemoryMapping.pdf

    AUTOSAR_SWS_BSW_Scheduler.pdf

[14] Basic Software Module Description Template
    AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[15] List of Basis Software Modules
    AUTOSAR_TR_BSWModuleList.pdf
3.2 Related standards and norms

[17] ISO11898 – Road vehicles - Controller area network (CAN)

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [16] (SWS BSW General), which is also valid for CAN Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for CAN Driver.
4 Constraints and assumptions

4.1 Limitations

A CAN controller always corresponds to one physical channel. It is allowed to connect physical channels on bus side. Regardless the CanIf module will treat the concerned CAN controllers separately.
A few CAN hardware units support the possibility to combine several CAN controllers by using the CAN RAM, to extend the number of message objects for one CAN controller. These combined CAN controller are handled as one controller by the Can module.
The Can module does not support CAN remote frames.

[SWS_Can_00237] [ The Can module shall not transmit messages triggered by remote transmission requests.] (SRS_Can_01147)

[SWS_Can_00236] [ The Can module shall initialize the CAN HW to ignore any remote transmission requests.] (SRS_Can_01147)

4.2 Applicability to car domains

The Can module can be used for any application, where the CAN protocol is used.
5 Dependencies to other modules

5.1 Static Configuration

The configuration elements described in chapter 10 can be referenced by other BSW modules for their configuration.

5.2 Driver Services

[SWS_Can_00238] [If the CAN controller is on-chip, the Can module shall not use any service of other drivers.] (SRS_BSW_00005)

[SWS_Can_00239] [The function Can_Init shall initialize all on-chip hardware resources that are used by the CAN controller. The only exception to this is the digital I/O pin configuration (of pins used by CAN), which is done by the port driver.] (SRS_BSW_00377)

[SWS_Can_00240] [The Mcu module (SPAL see [8]) shall configure register settings that are 'shared' with other modules.] ()

Implementation hint: The Mcu module shall be initialized before initializing the Can module.

[SWS_Can_00242] [If an off-chip CAN controller is used, the Can module shall use services of other MCAL drivers (e.g. SPI).] (SRS_BSW_00005)

Implementation hint: If the Can module uses services of other MCAL drivers (e.g. SPI), it must be ensured that these drivers are up and running before initializing the Can module.

The sequence of initialization of different drivers is partly specified in [7].

[SWS_Can_00244] [The Can module shall use the synchronous APIs of the underlying MCAL drivers and shall not provide callback functions that can be called by the MCAL drivers.] ()

Thus the type of connection between µC and CAN Hardware Unit has only impact on implementation and not on the API.

5.3 System Services

[SWS_Can_00280] [In special hardware cases, the Can module shall poll for events of the hardware.] ()

[SWS_Can_00281] [The Can module shall use the OsCounter provided by the system service for timeout detection in case the hardware does not react in the expected time (hardware malfunction) to prevent endless loops.] ()

Implementation hint: The blocking time of the Can module function that is waiting for hardware reaction shall be shorter than the CAN main function (i.e. Can_MainFunction_Read) trigger period, because the CAN main functions can’t be used for that purpose.

---

3 In this case the CAN driver is not any more part of the µC abstraction layer but put part of the ECU abstraction layer. Therefore it is (theoretically) allowed to use any µC abstraction layer driver it needs.
5.4 Can module Users

[SWS_Can_00058] The Can module interacts among other modules (eg. Diagnostic Event Manager (DEM), Default Error Tracer (DET), Ecu State Manager (ECUM)) with the CanIf module in a direct way. This document never specifies the actual origin of a request or the actual destination of a notification. The driver only sees the CanIf module as origin and destination.] (SRS_SPAL_12092)
5.5 File structure

5.5.1 Header file structure

[SWS_Can_00034] The header file structure for the Can module (SRS_BSW_00381, SRS_BSW_00412, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00348, SRS_BSW_00301)

[SWS_Can_00435] The Can.h file shall include Can_GeneralTypes.h (SRS_BSW_00381, SRS_BSW_00412, SRS_BSW_00346, SRS_BSW_00158, SRS_BSW_00348, SRS_BSW_00301)

[SWS_Can_00436] Can_GeneralTypes.h shall contain all types and constants that are shared among the AUTOSAR CAN modules Can, CanIf and CanTrcv.

[SWS_Can_00388] The header file Can.h shall include the header file ComStack_Types.h.
The CAN module does not provide callback functions (no Can_Cbk.h, see also SWS_Can_00244).}

The CAN module shall include the header file EcuM_Cbk.h, in which the callback functions called by the CAN module at the Ecu State Manager module are declared.

Can module implementations for off-chip CAN controllers shall include the header file Spi.h. By this inclusion, the APIs to access an external CAN controller by the SPI module [11] are included.

The CAN module shall include the header file Os.h file. By this inclusion, the API to read a OsCounter value (GetCounterValue) provided by the system service shall be included.
## 6 Requirements traceability

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Satisfied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRS_BSW_00005</td>
<td>Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces</td>
<td>SWS_Can_00238, SWS_Can_00242</td>
</tr>
<tr>
<td>SRS_BSW_00007</td>
<td>All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.</td>
<td>SWS_Can_00079</td>
</tr>
<tr>
<td>SRS_BSW_00101</td>
<td>The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function</td>
<td>SWS_Can_00250</td>
</tr>
<tr>
<td>SRS_BSW_00158</td>
<td>All modules of the AUTOSAR Basic Software shall strictly separate configuration from implementation</td>
<td>SWS_Can_00034</td>
</tr>
<tr>
<td>SRS_BSW_00159</td>
<td>All modules of the AUTOSAR Basic Software shall support a tool based configuration</td>
<td>SWS_Can_00022</td>
</tr>
<tr>
<td>SRS_BSW_00162</td>
<td>The AUTOSAR Basic Software shall provide a hardware abstraction layer</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00164</td>
<td>The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules</td>
<td>SWS_Can_00033</td>
</tr>
<tr>
<td>SRS_BSW_00167</td>
<td>All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks</td>
<td>SWS_Can_00024</td>
</tr>
<tr>
<td>SRS_BSW_00168</td>
<td>SW components shall be tested by a function defined in a common API in the Basis-SW</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00170</td>
<td>The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00301</td>
<td>All AUTOSAR Basic Software Modules shall only import the necessary information</td>
<td>SWS_Can_00034</td>
</tr>
<tr>
<td>SRS_BSW_00306</td>
<td>AUTOSAR Basic Software Modules shall be compiler and platform independent</td>
<td>SWS_Can_00079</td>
</tr>
<tr>
<td>SRS_BSW_00307</td>
<td>Global variables naming convention</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00308</td>
<td>AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file</td>
<td>SWS_Can_00079</td>
</tr>
<tr>
<td>SRS_BSW_00309</td>
<td>All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword</td>
<td>SWS_Can_00079</td>
</tr>
<tr>
<td>SRS_BSW_00312</td>
<td>Shared code shall be reentrant</td>
<td>SWS_Can_00214, SWS_Can_00231, SWS_Can_00232, SWS_Can_00233</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00323</td>
<td>All AUTOSAR Basic Software Modules shall check passed API parameters for validity</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00325</td>
<td>The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00330</td>
<td>It shall be allowed to use macros instead of functions where source code is used and runtime is critical</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00331</td>
<td>All Basic Software Modules shall strictly separate error and status information</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00336</td>
<td>Basic SW module shall be able to shutdown</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00337</td>
<td>Classification of development errors</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00342</td>
<td>It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00344</td>
<td>BSW Modules shall support link-time configuration</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00346</td>
<td>All AUTOSAR Basic Software Modules shall provide at least a basic set of module files</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00347</td>
<td>A Naming separation of different instances of BSW drivers shall be in place</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00348</td>
<td>All AUTOSAR standard types and constants shall be placed and organized in a standard type header file</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00353</td>
<td>All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00357</td>
<td>For success/failure of an API call a standard return type shall be defined</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00358</td>
<td>The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00359</td>
<td>All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00361</td>
<td>All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00369</td>
<td>All AUTOSAR Basic Software Modules shall not return specific development error codes via the API</td>
<td></td>
</tr>
<tr>
<td>SRS_BSW_00373</td>
<td>The main processing function of each</td>
<td></td>
</tr>
<tr>
<td>Requirement ID</td>
<td>Description</td>
<td>Dependencies</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>SRS_BSW_00375</td>
<td>Basic Software Modules shall report wake-up reasons</td>
<td>SWS_Can_00271, SWS_Can_00364</td>
</tr>
<tr>
<td>SRS_BSW_00377</td>
<td>A Basic Software Module can return a module specific types</td>
<td>SWS_Can_00239</td>
</tr>
<tr>
<td>SRS_BSW_00378</td>
<td>AUTOSAR shall provide a boolean type</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00381</td>
<td>The pre-compile time parameters shall be placed into a separate configuration header file</td>
<td>SWS_Can_00034</td>
</tr>
<tr>
<td>SRS_BSW_00383</td>
<td>The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00385</td>
<td>List possible error notifications</td>
<td>SWS_Can_00104</td>
</tr>
<tr>
<td>SRS_BSW_00386</td>
<td>The BSW shall specify the configuration for detecting an error</td>
<td>SWS_Can_00089</td>
</tr>
<tr>
<td>SRS_BSW_00395</td>
<td>The Basic Software Module specifications shall list all configuration parameter dependencies</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00397</td>
<td>The configuration parameters in pre-compile time are fixed before compilation starts</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00398</td>
<td>The link-time configuration is achieved on object code basis in the stage after compiling and before linking</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00399</td>
<td>Parameter-sets shall be located in a separate segment and shall be loaded after the code</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00400</td>
<td>Parameter shall be selected from multiple sets of parameters after code has been loaded and started</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00404</td>
<td>BSW Modules shall support post-build configuration</td>
<td>SWS_Can_00021</td>
</tr>
<tr>
<td>SRS_BSW_00405</td>
<td>BSW Modules shall support multiple configuration sets</td>
<td>SWS_Can_00021</td>
</tr>
<tr>
<td>SRS_BSW_00406</td>
<td>A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called</td>
<td>SWS_Can_00103, SWS_Can_91005, SWS_Can_91016</td>
</tr>
<tr>
<td>SRS_BSW_00409</td>
<td>All production code error ID symbols are defined by the Dem module and shall be retrieved by the other BSW modules from Dem configuration</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00412</td>
<td>References to c-configuration parameters shall be placed into a separate h-file</td>
<td>SWS_Can_00034</td>
</tr>
<tr>
<td>SRS_BSW_00413</td>
<td>An index-based accessing of the instances of BSW modules shall be done</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00414</td>
<td>Init functions shall have a pointer to a configuration structure as single parameter</td>
<td>SWS_Can_00223</td>
</tr>
<tr>
<td>SRS_BSW_00415</td>
<td>Interfaces which are provided exclusively for one module shall be separated into a dedicated header file</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00416</td>
<td>The sequence of modules to be initialized shall be configurable</td>
<td>SWS_Can_91005, SWS_Can_91016</td>
</tr>
<tr>
<td>SRS_BSW_00417</td>
<td>Software which is not part of the SW-C shall report error events only after the DEM is fully operational.</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00422</td>
<td>Pre-de-bouncing of error status information is done within the DEM</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00423</td>
<td>BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00424</td>
<td>BSW module main processing functions shall not be allowed to enter a wait state</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00425</td>
<td>The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00426</td>
<td>BSW Modules shall ensure data consistency of data which is shared between BSW modules</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00427</td>
<td>ISR functions shall be defined and documented in the BSW module description template</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00428</td>
<td>A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence</td>
<td>SWS_Can_00110</td>
</tr>
<tr>
<td>SRS_BSW_00429</td>
<td>Access to OS is restricted</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00432</td>
<td>Modules should have separate main processing functions for read/receive and write/transmit data path</td>
<td>SWS_Can_00031, SWS_Can_00108, SWS_Can_00112</td>
</tr>
<tr>
<td>SRS_BSW_00433</td>
<td>Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00438</td>
<td>Configuration data shall be defined in a structure</td>
<td>SWS_Can_00291</td>
</tr>
<tr>
<td>SRS_BSW_00439</td>
<td>Enable BSW modules to handle interrupts</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00440</td>
<td>The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00447</td>
<td>Standardizing Include file structure of BSW Modules Implementing Autosar Service</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00449</td>
<td>BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType</td>
<td>SWS_CAN_00506, SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_BSW_00453</td>
<td>BSW Modules shall be harmonized</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_Can_01005</td>
<td>The CAN Interface shall perform a check for correct DLC of received PDUs</td>
<td>SWS_Can_00218</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01041</td>
<td>The CAN Driver shall implement an interface for initialization</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01042</td>
<td>The CAN Driver shall support dynamic selection of configuration sets</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01043</td>
<td>The CAN Driver shall provide a service to enable/disable interrupts of the CAN Controller.</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01045</td>
<td>The CAN Driver shall offer a reception indication service.</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01049</td>
<td>The CAN Driver shall provide a dynamic transmission request service</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01051</td>
<td>The CAN Driver shall provide a transmission confirmation service</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01053</td>
<td>The CAN Driver shall provide a service to change the CAN controller mode.</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01054</td>
<td>The CAN Driver shall provide a notification for controller wake-up events</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01055</td>
<td>The CAN Driver shall provide a notification for bus-off state</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01059</td>
<td>The CAN Driver shall guarantee data consistency of received L-PDUs</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01060</td>
<td>The CAN driver shall not recover from bus-off automatically</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01062</td>
<td>Each event for each CAN Controller shall be configurable to be detected by polling or by an interrupt</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01122</td>
<td>The CAN driver shall support the situation where a wake up by bus occurs during the same time the transition to standby/sleep is in progress</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01125</td>
<td>The CAN stack shall ensure not to lose messages in receive direction</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01126</td>
<td>The CAN stack shall be able to produce 100% bus load</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01130</td>
<td>Receive Status Interface of CAN Interface</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01132</td>
<td>The CAN driver shall be able to detect notification events message object specific by CAN-Interrupt and polling</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01134</td>
<td>The CAN Driver shall support multiplexed transmission</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01135</td>
<td>It shall be possible to configure one or several TX Hardware Objects</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01139</td>
<td>The CAN Interface and Driver shall offer a CAN Controller specific interface for</td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SRS_Can_01147</td>
<td>The CAN Driver shall not support remote frames</td>
<td>SWS_Can_00236, SWS_Can_00237</td>
</tr>
<tr>
<td>SRS_Can_01160</td>
<td>Padding of bytes due to discrete CAN FD DLC</td>
<td>SWS_CAN_00502</td>
</tr>
<tr>
<td>SRS_Can_01162</td>
<td>The CAN Interface shall support classic CAN and CAN FD frames</td>
<td>SWS_CAN_00501</td>
</tr>
<tr>
<td>SRS_Can_01166</td>
<td>The CAN Driver shall implement an interface for de-initialization</td>
<td>noname, SWS_Can_91002, SWS_Can_91010</td>
</tr>
<tr>
<td>SRS_CAN_01167</td>
<td>-</td>
<td>SWS_Can_91008</td>
</tr>
<tr>
<td>SRS_SPAL_00157</td>
<td>All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers</td>
<td>SWS_Can_00026, SWS_Can_00031, SWS_Can_00108, SWS_Can_00112</td>
</tr>
<tr>
<td>SRS_SPAL_12056</td>
<td>All driver modules shall allow the static configuration of notification mechanism</td>
<td>SWS_Can_00235</td>
</tr>
<tr>
<td>SRS_SPAL_12057</td>
<td>All driver modules shall implement an interface for initialization</td>
<td>SWS_Can_00245, SWS_Can_00246</td>
</tr>
<tr>
<td>SRS_SPAL_12063</td>
<td>All driver modules shall only support raw value mode</td>
<td>SWS_Can_00059, SWS_Can_00060</td>
</tr>
<tr>
<td>SRS_SPAL_12064</td>
<td>All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_SPAL_12067</td>
<td>All driver modules shall set their wake-up conditions depending on the selected operation mode</td>
<td>SWS_Can_00257</td>
</tr>
<tr>
<td>SRS_SPAL_12068</td>
<td>The modules of the MCAL shall be initialized in a defined sequence</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_SPAL_12069</td>
<td>All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason</td>
<td>SWS_Can_00271, SWS_Can_00364</td>
</tr>
<tr>
<td>SRS_SPAL_12075</td>
<td>All drivers with random streaming capabilities shall use application buffers</td>
<td>SWS_Can_00011</td>
</tr>
<tr>
<td>SRS_SPAL_12077</td>
<td>All drivers shall provide a non blocking implementation</td>
<td>SWS_Can_00372</td>
</tr>
<tr>
<td>SRS_SPAL_12092</td>
<td>The driver's API shall be accessed by its handler or manager</td>
<td>SWS_Can_00058</td>
</tr>
<tr>
<td>SRS_SPAL_12125</td>
<td>All driver modules shall only initialize the configured resources</td>
<td>SWS_Can_00053</td>
</tr>
<tr>
<td>SRS_SPAL_12129</td>
<td>The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function</td>
<td>SWS_Can_00033</td>
</tr>
<tr>
<td>SRS_SPAL_12163</td>
<td>All driver modules shall implement an interface for de-initialization</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_SPAL_12169</td>
<td>All driver modules that provide different operation modes shall provide a service for mode selection</td>
<td>SWS_Can_00017</td>
</tr>
<tr>
<td>SRS_SPAL_12263</td>
<td>The implementation of all driver modules</td>
<td>SWS_Can_00021</td>
</tr>
<tr>
<td>Specification</td>
<td>Configuration</td>
<td>Reference Codes</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>shall allow the configuration of specific module parameter types at link time</td>
<td>Configuration data shall be kept constant</td>
<td>SWS_Can_00021</td>
</tr>
<tr>
<td>SRS_SPAL_12265</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All driver modules shall have a specific behavior after a development error detection</td>
<td>All driver modules shall have a specific behavior after a development error detection</td>
<td>SWS_Can_00089, SWS_Can_00091</td>
</tr>
<tr>
<td>SRS_SPAL_12448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific rules regarding initialization of controller registers shall apply to all driver implementations</td>
<td>Specific rules regarding initialization of controller registers shall apply to all driver implementations</td>
<td>SWS_Can_00407</td>
</tr>
<tr>
<td>SRS_SPAL_12461</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The register initialization settings shall be published</td>
<td>The register initialization settings shall be published</td>
<td>SWS_Can_00999</td>
</tr>
<tr>
<td>SRS_SPAL_12462</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The register initialization settings shall be combined and forwarded</td>
<td>The register initialization settings shall be combined and forwarded</td>
<td>SWS_Can_00024</td>
</tr>
</tbody>
</table>


7 Functional specification

On L-PDU transmission, the Can module writes the L-PDU in an appropriate buffer inside the CAN controller hardware. See chapter 7.5 for closer description of L-PDU transmission.

On L-PDU reception, the Can module calls the RX indication callback function with ID, Data Length and pointer to L-SDU as parameter. See chapter 7.6 for closer description of L-PDU reception.

The Can module provides an interface that serves as periodical processing function, and which must be called by the Basic Software Scheduler module periodically. Furthermore, the Can module provides services to control the state of the CAN controllers. Bus-off and Wake-up events are notified by means of callback functions.

The Can module is a Basic Software Module that accesses hardware resources. Therefore, it is designed to fulfill the requirements for Basic Software Modules specified in AUTOSAR_SRS_SPAL (see [3]).

[SWS_Can_00033] The Can module shall implement the interrupt service routines for all CAN Hardware Unit interrupts that are needed. (SRS_BSW_00164, SRS_SPAL_12129)

[SWS_Can_00419] The Can module shall disable all unused interrupts in the CAN controller. ()

[SWS_Can_00420] The Can module shall reset the interrupt flag at the end of the ISR (if not done automatically by hardware). ()

Implementation hint: The Can module shall not set the configuration (i.e. priority) of the vector table entry.

[SWS_Can_00079] The Can module shall fulfill all design and implementation guidelines described in [2]. (SRS_BSW_00007, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00330)

7.1 Driver scope

One Can module provides access to one CAN Hardware Unit that may consist of several CAN controllers.

[SWS_Can_00077] For CAN Hardware Units of different type, different Can modules shall be implemented. (SRS_BSW_00347)

[SWS_Can_00284] In case several CAN Hardware Units (of same or different vendor) are implemented in one ECU the function names, and global variables of the Can modules shall be implemented such that no two functions with the same name are generated. ()

The naming convention is as follows:

<Can module name>_<vendorID>_<Vendor specific API name><driver abbreviation>()

SRS_BSW_00347 specifies the naming convention.

[SWS_Can_00385] The naming conventions shall be used only in that case, if multiple different CAN controller types on one ECU have to be supported. ()

[SWS_Can_00386] If only one controller type is used, the original naming conventions without any <driver abbreviation> extensions are sufficient. ()

See [5] for description how several Can modules are handled by the CanIf module.
7.2 Driver State Machine

The Can module has a very simple state machine, with the two states CAN_UNINIT and CAN_READY. Figure 7.1 shows the state machine.

[SWS_Can_00103] After power-up/reset, the Can module shall be in the state CAN_UNINIT. (SRS_BSW_00406)

Figure 7-1

[SWS_Can_00246] The function Can_Init shall change the module state to CAN_READY, after initializing all controllers inside the HW Unit. (SRS_SPAL_12057, SRS_Can_01041)

[SWS_Can_00245] The function Can_Init shall initialize all CAN controllers according to their configuration. (SRS_SPAL_12057, SRS_Can_01041)

Each CAN controller must then be started separately by calling the function Can_SetControllerMode(CAN_CS_STARTED). Implementation hint:

Hardware register settings that have impact on all CAN controllers inside the HW Unit can only be set in the function Can_Init.

Implementation hint:

The ECU State Manager module shall call Can_Init at most once during runtime.

[SWS_Can_91009] The function Can_DeInit shall change the module state to CAN_UNINIT before de-initializing all controllers inside the HW unit. (SRS_Can_01166) Refer to [SWS_Can_91010]

7.3 CAN Controller State Machine

Each CAN controller has complex state machines implemented in hardware. For simplification, the number of states is reduced to the following four basic states in this description: UNINIT, STOPPED, STARTED and SLEEP.

Any CAN hardware access is encapsulated by functions of the Can module, but the Can module does not memorize the state changes.
The Can module offers the services Can_Init, Can_SetBaudrate and Can_SetControllerMode. These services perform the necessary register settings that cause the required change of the hardware CAN controller state.

There are two possibilities for triggering state changes by external events:
- Bus-off event
- HW wakeup event

These events are indicated either by an interrupt or by a status bit that is polled in the Can_MainFunction_BusOff or Can_MainFunction_Wakeup.

The Can module does the register settings that are necessary to fulfill the required behavior (i.e. no hardware recovery in case of bus off).

Then it notifies the CanIf module with the corresponding callback function. The software state is then changed inside this callback function.

In case development errors are enabled and there is a not allowed transition requested by the upper layer, the Can module shall rise the development error CAN_E_TRANSITION.

The Can module does not check the actual state before it performs Can_Write or raises callbacks.

### 7.3.1 CAN Controller State Description

This chapter describes the required hardware behavior for the different controller states.

**CAN controller state UNINIT**

The CAN controller is not initialized. All registers belonging to the CAN module are in reset state, CAN interrupts are disabled. The CAN Controller is not participating on the CAN bus.

**CAN controller state STOPPED**

In this state the CAN Controller is initialized but does not participate on the bus. In addition, error frames and acknowledges must not be sent.

(Example: For many controllers entering an ‘initialization’-mode causes the controller to be stopped.)

**CAN controller state STARTED**

The controller is in a normal operation mode with complete functionality, which means it participates in the network. For many controllers leaving the ‘initialization’-mode causes the controller to be started.

**CAN controller state SLEEP**
The hardware settings only differ from state STOPPED for CAN hardware that support a sleep mode (wake-up over CAN bus directly supported by CAN hardware).

[SWS_Can_00257] When the CAN hardware supports sleep mode and is triggered to transition into SLEEP state, the Can module shall set the controller to the SLEEP state from which the hardware can be woken over CAN Bus.] (SRS_SPAL_12067)

[SWS_Can_00258] When the CAN hardware does not support sleep mode and is triggered to transition into SLEEP state, the Can module shall emulate a logical SLEEP state from which it returns only, when it is triggered by software to transition into STOPPED state.] ()

[SWS_Can_00404] The CAN hardware shall remain in state STOPPED, while the logical SLEEP state is active.] ()

7.3.2 CAN Controller State Transitions

A state transition is triggered by software with the function Can_SetControllerMode with the required transition as parameter. A successful state transition triggered by software is notified by the callback function (CanIf_ControllerModeIndication). The monitoring whether the requested state is achieved is part of an upper layer module and is not part of the Can module.

Some transitions are triggered by events on the bus (hardware). These transitions cause a notification by means of a callback function (CanIf_ControllerBusOff, EcuM_CheckWakeup).

The behavior for invalid transitions in production code is undefined. Figure 7-2 shows all valid state transitions.
7.3.3 State transition caused by function Can_Init

- UNINIT → STOPPED (for all controllers in HW unit)
- software triggered by the function call Can_Init
- does configuration for all CAN controllers inside HW Unit

All control registers are set according to the static configuration.

[SWS_Can_00259] The function Can_Init shall set all CAN controllers in the state STOPPED.]

When the function Can_Init is entered and the Can module is not in state CAN_UNINIT or the CAN controllers are not in state UNINIT, it shall raise the error CAN_E_TRANSITION (Compare to SWS_Can_00174 and SWS_Can_00408).
7.3.4 State transition caused by function Can_SetBaudrate

- STOPPED -> STOPPED; SLEEP -> SLEEP; STARTED -> STARTED
- software triggered by the function call Can_SetBaudrate
- changes the CAN controller configuration

CAN controller registers are set according to the static configurations.

[SWS_Can_00256]  
If the call of Can_SetBaudrate() would cause a re-initialization of the CAN Controller and the CAN Controller is not in state STOPPED, it shall return E_NOT_OK. ()

[SWS_Can_00260]  
If re-initialization is necessary the function Can_SetBaudrate shall maintain the CAN controller in the state STOPPED. ()

[SWS_Can_00422]  
If re-initialization is necessary the function Can_SetBaudrate shall ensure that any settings that will cause the CAN controller to participate in the network are not set. ()

7.3.5 State transition caused by function Can_SetControllerMode

The software can trigger a CAN controller state transition with the function Can_SetControllerMode. Depending on the CAN hardware, a change of a register setting to transition to a new CAN controller state may take over only after a delay. The Can module notifies the upper layer (CanIf_ControllerModeIndication) after a successful state transition about the new state. The monitoring whether the requested state is achieved is part of an upper layer module and is not part of the Can module.

[SWS_Can_00370]  
The function Can_Mainfunction_Mode shall poll a flag of the CAN status register until the flag signals that the change takes effect and notify the upper layer with function CanIf_ControllerModeIndication about a successful state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId. ()

[SWS_Can_00398]  
The function Can_SetControllerMode shall use the system service GetCounterValue for timeout monitoring to avoid blocking functions. ()

[SWS_Can_00372]  
In case the flag signals that the change takes no effect and the maximum time CanTimeoutDuration is elapsed, the function Can_SetControllerMode shall be left and the function Can_Mainfunction_Mode shall continue to poll the flag. (SRS_SPAL_12077)

[SWS_Can_00373]  
The function Can_Mainfunction_Mode shall call the function CanIf_ControllerModeIndication to notify the upper layer about a successful state transition of the corresponding CAN controller referred by abstract CanIf ControllerId, in case the state transition was triggered by function Can_SetControllerMode. ()

State transition caused by function Can_SetControllerMode  
(CAN_CS_STARTED)
STOPPED ➔ STARTED
software triggered

[SWS_Can_00261]  | The function Can_SetControllerMode(CAN_CS_STARTED) shall set the hardware registers in a way that makes the CAN controller participating on the network.]  ()

[SWS_Can_00262]  | The function Can_SetControllerMode(CAN_CS_STARTED) shall wait for limited time until the CAN controller is fully operational. Compare to SWS_Can_00398.]  ()

Transmit requests that are initiated before the CAN controller is operational get lost. The only indicator for operability is the reception of TX confirmations or RX indications. The sending entities might get a confirmation timeout and need to be able to cope with that.

[SWS_Can_00409]  | When the function Can_SetControllerMode (CAN_CS_STARTED) is entered and the CAN controller is not in state STOPPED it shall detect a invalid state transition (Compare to SWS_Can_00200).]  ()

State transition caused by function Can_SetControllerMode (CAN_CS_STOPPED)

- STARTED ➔ STOPPED
- SLEEP ➔ STOPPED
- software triggered

[SWS_Can_00263]  | The function Can_SetControllerMode(CAN_CS_STOPPED) shall set the bits inside the CAN hardware such that the CAN controller stops participating on the network.]  ()

[SWS_Can_00264]  | The function Can_SetControllerMode(CAN_CS_STOPPED) shall wait for a limited time until the CAN controller is really switched off. Compare to SWS_Can_00398.]  ()

[SWS_Can_00267]  | If the CAN HW does not support a sleep mode, the transition from SLEEP to STOPPED shall return from the logical sleep mode, but have no effect to the CAN controller state (as the controller is already in stopped state).]  ()

[SWS_Can_00268]  | The function Can_SetControllerMode(CAN_CS_STOPPED) shall wait for a limited time until the CAN controller is in STOPPED state. Compare to SWS_Can_00398.]  ()

[SWS_Can_00262]  | The function Can_SetControllerMode(CAN_CS_STOPPED) shall cancel pending messages. ]  ()

State transition caused by function Can_SetControllerMode(CAN_CS_SLEEP)

- STOPPED ➔ SLEEP
- software triggered

[SWS_Can_00265]  | The function Can_SetControllerMode(CAN_CS_SLEEP) shall set the controller into sleep mode.]  ()
If the CAN HW does support a sleep mode, the function \texttt{Can\_SetControllerMode(CAN\_CS\_SLEEP)} shall wait for a limited time until the CAN controller is in SLEEP state and it is assured that the CAN hardware is wake able. Compare to \texttt{SWS\_Can\_00398}.

If the CAN HW does not support a sleep mode, the function \texttt{Can\_SetControllerMode(CAN\_CS\_SLEEP)} shall set the CAN controller to the logical sleep mode.

This logical sleep mode shall left only, if function \texttt{Can\_SetControllerMode(CAN\_CS\_STOPPED)} is called.

When the function \texttt{Can\_SetControllerMode(CAN\_CS\_SLEEP)} is entered and the CAN controller is neither in state STOPPED nor in state SLEEP, it shall detect a invalid state transition (Compare to \texttt{SWS\_Can\_00200}).

### 7.3.6 State transition caused by Hardware Events

#### State transition caused by Hardware Wakeup (triggered by wake-up event from CAN bus)

- SLEEP $\rightarrow$ STOPPED
- triggered by incoming L-PDUs
- The ECU Statemanager module is notified with the function \texttt{EcuM\_CheckWakeup}

This state transition will only occur when sleep mode is supported by hardware.

On hardware wakeup (triggered by a wake-up event from CAN bus), the CAN controller shall transition into the state STOPPED.

On hardware wakeup (triggered by a wake-up event from CAN bus), the Can module shall call the function \texttt{EcuM\_CheckWakeup} either in interrupt context or in the context of \texttt{Can\_MainFunction\_Wakeup}.

The Can module shall not further process the L-PDU that caused a wake-up.

In case of a CAN bus wake-up during sleep transition, the function \texttt{Can\_SetControllerMode(CAN\_CS\_STOPPED)} shall return \texttt{E\_NOT\_OK}.

#### State transition caused by Bus-Off (triggered by state change of CAN controller)

- STARTED $\rightarrow$ STOPPED
- triggered by hardware if the CAN controller reaches bus-off state
- The CanIf module is notified with the function \texttt{CanIf\_ControllerBusOff} after STOPPED state is reached referring to the corresponding CAN controller with the abstract \texttt{CanIf\_ControllerId}.

(SRS\_Can\_01055)
7.3.7 State transition caused by function Can_DeInit

- STOPPED -> UNINIT; SLEEP -> UNINIT (for all controllers in HW unit)
- software triggered by the function call Can_DeInit
- prepares all CAN controllers inside HW Unit to be re-configured

[SWS_Can_91010] [The function Can_DeInit shall set all CAN controllers in the state UNINIT] (SRS_Can_01166, SRS_Can_01053)

When the function Can_DeInit is entered and the Can module is not in state CAN_READY or any of the CAN controllers is in state STARTED, it shall raise the error CAN_E_TRANSITION (Refer to [SWS_Can_91011] and [SWS_Can_91012]).

7.4 Can module/Controller Initialization

The ECU State Manager module shall initialize the Can module during startup phase by calling the function Can_Init before using any other functions of the Can module.

[SWS_Can_00250] [The function Can_Init shall initialize:

- static variables, including flags,
- Common setting for the complete CAN HW unit
- CAN controller specific settings for each CAN controller] (SRS_BSW_00101)

[SWS_Can_00053] [Can_Init shall not change registers of CAN controller Hardware resources that are not used.] (SRS_SPAL_12125)

The Can module shall apply the following rules regarding initialization of controller registers:

- [SWS_Can_00407] [If the hardware allows for only one usage of the register, the Can module implementing that functionality is responsible initializing the register.
- If the register can affect several hardware modules and if it is an I/O register it shall be initialized by the PORT driver.
- If the register can affect several hardware modules and if it is not an I/O register it shall be initialized by the MCU driver.
- One-time writable registers that require initialization directly after reset shall be initialized by the startup code.
- All other registers shall be initialized by the startup code. (SRS_SPAL_12461)

[SWS_Can_00056] Post-Build configuration elements that are marked as ‘multiple’ ('M' or 'x') in chapter 10 can be selected by passing the pointer ‘Config’ to the init function of the module. ()

[SWS_Can_00062] If Can_SetBaudrate determines that the aimed configuration change requires a re-initialization and the CAN Controller is in STOPPED, the function Can_SetBaudrate shall re-initialize the CAN controller and the controller specific settings. (SRS_Can_01139, SRS_Can_01042)

If re-initialization is necessary, the CAN Controller has to be switched to STOPPED before Can_SetBaudrate() can be executed and the new baud rate configuration can be applied.

[SWS_Can_00255] The function Can_SetBaudrate shall only affect register areas that contain specific configuration for a single CAN controller. ()

[SWS_Can_00021] The desired CAN controller configuration can be selected with the parameter Config. (SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_SPAL_12263, SRS_SPAL_12265)

[SWS_Can_00291] Config is a pointer into an array of implementation specific data structure stored in ROM. The different controller configuration sets are located as data structures in ROM. (SRS_BSW_00438)

The possible values for Config are provided by the configuration description (see chapter 10).

The Can module configuration defines the global CAN HW Unit settings and references to the default CAN controller configuration sets.

### 7.5 L-PDU transmission

On L-PDU transmission, the Can module converts the L-PDU contents ID and Data Length to a hardware specific format (if necessary) and triggers the transmission.

[SWS_Can_00059] Data mapping by CAN to memory is defined in a way that the CAN data byte which is sent out first is array element 0, the CAN data byte which is sent out last is array element 7 or 63 in case of CAN FD. (SRS_SPAL_12063)

[SWS_Can_00427] If the presentation inside the CAN Hardware buffer differs from AUTOSAR definition, the Can module must provide an adapted SDU-Buffer for the upper layers. ()

[SWS_Can_00100] Several TX hardware objects with unique HTHs may be configured. The CanIf module provides the HTH as parameter of the TX request. See Figure 7-3 for a possible configuration. (SRS_Can_01135)
Figure 7-3: Example of assignment of HTHs and HRHs to the Hardware Objects. The numbering of HTHs and HRHs are implementation specific. The chosen numbering is only an example.

[SWS_Can_00276]  The function Can_Write shall store the swPduHandle that is given inside the parameter PduInfo until the Can module calls the CanIf_TxConfirmation for this request where the swPduHandle is given as parameter. ()
The feature of SWS_Can_00276 is used to reduce time for searching in the CanIf module implementation.

[SWS_Can_00016]  The Can module shall call CanIf_TxConfirmation to indicate a successful transmission. It shall either called by the TX-interrupt service routine of the corresponding HW resource or inside the Can_MainFunction_Write in case of polling mode.] (SRS_Can_01051)

7.5.1 Priority Inversion

Multiplexed transmission is necessary to prevent priority inversion (see chapter 2.1).

[SWS_Can_00277]  The Can module shall allow that the functionality “Multiplexed Transmission” is statically configurable (ON | OFF) at pre-compile time.] (SRS_Can_01134)

[SWS_Can_00401]  Several transmit hardware objects (defined by “CanHwObjectCount”) shall be assigned by one HTH to represent one transmit entity to the upper layer.] (SRS_Can_01134)

[SWS_Can_00402]  The Can module shall support multiplexed transmission mechanisms for devices where either

- Multiple transmit hardware objects, which are grouped to a transmit entity can be filled over the same register set, and the microcontroller stores the L-PDU into a free buffer autonomously,

or
The Hardware provides registers or functions to identify a free transmit hardware object within a transmit entity.\footnote{SRS_Can_01134}

\[\text{SWS\_Can\_00403}\] The Can module shall support multiplexed transmission for devices, which send L-PDUs in order of L-PDU priority.\footnote{SRS_Can_01134}

Note: Software emulation of priority handling should be avoided, because the overhead would void the advantage of the multiplexed transmission.

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
HRH - 0 & ID & DLC & SDU \\
\hline
HRH - 1 & ID & DLC & SDU \\
\hline
unused & ID & DLC & SDU \\
\hline
HRH - 2 & ID & DLC & SDU \\
\hline
HRH - 3 & ID & DLC & SDU \\
\hline
HTH - 4 & ID & DLC & SDU \\
\hline
\end{tabular}
\end{center}

\textbf{Figure 7-4}: Example of assignment of HTHs and HRHs to the Hardware Objects with multiplexed transmission. The numbering of HTHs and HRHs are implementation specific. The chosen numbering is only an example.

\section*{7.5.2 Transmit Data Consistency}

\[\text{SWS\_Can\_00011}\] The Can module shall directly copy the data from the upper layer buffers. It is the responsibility of the upper layer to keep the buffer consistent until return of function call (Can_Write).\footnote{SRS_SPAL\_12075, SRS_Can\_01059}
7.6 L-PDU reception

[SWS_Can_00279] [On L-PDU reception, the Can module shall call the RX indication callback function CanIf_RxIndication with ID, Hoh, abstract CanIf ControllerId in parameter Mailbox, and the Data Length and pointer to the L-SDU buffer in parameter PduInfoPtr.] (SRS_Can_01045)

[SWS_Can_00423] [In case of an Extended CAN frame, the Can module shall convert the ID to a standardized format since the Upper layer (CANIF) does not know whether the received CAN frame is a Standard CAN frame or Extended CAN frame. In case of an Extended CAN frame, MSB of a received CAN frame ID needs to be made as ‘1’ to mark the received CAN frame as Extended.] ()

[SWS_Can_00396] [The RX-interrupt service routine of the corresponding HW resource or the function Can_MainFunction_Read in case of polling mode shall call the callback function CanIf_RxIndication.] (SRS_Can_01045)

[SWS_Can_00060] [Data mapping by CAN to memory is defined in a way that the CAN data byte which is received first is array element 0, the CAN data byte which is received last is array element 7 or 63 in case of CAN FD. If the presentation inside the CAN Hardware buffer differs from AUTOSAR definition, the Can module must provide an adapted SDU-Buffer for the upper layers.] (SRS_SPAL_12063)

[SWS_Can_00501] [CanDrv shall indicate whether the received message is a conventional CAN frame or a CAN FD frame as described in Can_IdType.] (SRS_Can_01162)

7.6.1 Receive Data Consistency

To prevent loss of received messages, some controllers support a FIFO built from a set of hardware objects, while on other controllers it is possible to configure another hardware object with the same properties that works as a shadow buffer and steps in when the main object is busy.

[SWS_Can_00489] [The CAN driver shall support controllers which implement a hardware FIFO. The size of the FIFO is configured via "CanHwObjectCount".] ()

[SWS_Can_00490] [Controllers that do not support a hardware FIFO often provide the capabilities to implement a shadow buffer mechanism, where additional hardware objects take over when the primary hardware object is busy. The number of hardware objects is configured via "CanHwObjectCount".] ()
**Message Objects of CAN Hardware**

<table>
<thead>
<tr>
<th>HRH = 0</th>
<th>ID</th>
<th>DLC</th>
<th>SDU</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRH = 1</td>
<td>ID</td>
<td>DLC</td>
<td>SDU</td>
</tr>
<tr>
<td>HRH = 2</td>
<td>ID</td>
<td>DLC</td>
<td>SDU</td>
</tr>
<tr>
<td>HRH = 3</td>
<td>ID</td>
<td>DLC</td>
<td>SDU</td>
</tr>
</tbody>
</table>

Figure 7-5: Example of assignment of same HRHs to multiple Hardware Objects. The chosen numbering is only an example.

[SWS_Can_00299] The Can module shall copy the L-SDU in a shadow buffer after reception, if the RX buffer cannot be protected (locked) by CAN Hardware against overwriting by a newly received message. ()

[SWS_Can_00300] The Can module shall copy the L-SDU in a shadow buffer, if the CAN Hardware is not globally accessible. ()

The complete RX processing (including copying to destination layer, e.g. COM) is done in the context of the RX interrupt or in the context of the Can_MainFunction_Read.

[SWS_Can_00012] The Can module shall guarantee that neither the ISRs nor the function Can_MainFunction_Read can be interrupted by itself. The CAN hardware (or shadow) buffer is always consistent, because it is written and read in sequence in exactly one function that is never interrupted by itself. (SRS_Can_01059)

If the CAN hardware cannot be configured to lock the RX hardware object after reception (hardware feature), it could happen that the hardware buffer is overwritten by a newly arrived message. In this case, the CAN controller detects an "overwrite" event, if supported by hardware.

If the CAN hardware can be configured to lock the RX hardware object after reception, it could happen that the newly arrived message cannot be stored to the hardware buffer. In this case, the CAN controller detects an "overrun" event, if supported by hardware.

[SWS_Can_00395] Can module shall raise the runtime error CAN_E_DATALOST in case of "overwrite" or "overrun" event detection. ()

Implementation Hint:
The system designer shall assure that the runtime for message reception (interrupt driven or polling) correlates with the fastest possible reception in the system.
7.7 Wakeup concept

The Can module handles wakeups that can be detected by the Can controller itself and not via the Can transceiver. There are two possible scenarios: wakeup by interrupt and wakeup by polling. For wakeup by interrupt, an ISR of the Can module is called when the hardware detects the wakeup.

[SWS_Can_00364] If the ISR for wakeup events is called, it shall call EcuM_CheckWakeup in turn. The parameter passed to EcuM_CheckWakeup shall be the ID of the wakeup source referenced by the CanWakeupSourceRef configuration parameter. (SRS_BSW_00375, SRS_SPAL_12069, SRS_Can_01054)

The ECU State Manager will then set up the MCU and call the Can module back via the Can Interface, resulting in a call to Can_CheckWakeup. When wakeup events are detected by polling, the ECU State Manager will cyclically call Can_CheckWakeup via the Can Interface as before. In both cases, Can_CheckWakeup will check if there was a wakeup detected by a Can controller and return the result. The CAN driver will then inform the ECU State Manager of the wakeup event via EcuM_SetWakeupEvent.

The wakeup validation to prevent false wakeup events, will be done by the ECU State Manager and the Can Interface afterwards and without any help from the Can module.

For a general description of the wakeup mechanisms and wakeup sequence diagrams refer to Specification of ECU State Manager [7].

7.8 Notification concept

The Can module offers only an event triggered notification interface to the CanIf module. Each notification is represented by a callback function.

[SWS_Can_00099] The hardware events may be detected by an interrupt or by polling status flags of the hardware objects. The configuration possibilities regarding polling is hardware dependent (i.e. which events can be polled, which events need to be polled), and not restricted by this standard. (SRS_Can_01132)

[SWS_Can_00007] It shall be possible to configure the driver such that no interrupts at all are used (complete polling). (SRS_Can_01062)

The configuration of what is and is not polled by the Can module is internal to the driver, and not visible outside the module. The polling is done inside the CAN main functions (Can_MainFunction_xxx). Also the polled events are notified by the appropriate callback function. Then the call context is not the ISR but the CAN main function. The implementation of all callback functions shall be done as if the call context was the ISR.

For further details see also description of the CAN main functions Can_MainFunction_Read, Can_MainFunction_Write, Can_MainFunction_BusOff and Can_MainFunction_Wakeup.
7.9 Reentrancy issues

A routine must satisfy the following conditions to be reentrant:

- It uses all shared variables in an atomic way, unless each is allocated to a specific instance of the function.
- It does not call non-reentrant functions.
- It does not use the hardware in a non-atomic way.

Transmit requests are simply forwarded by the CanIf module inside the function CanIf_Transmit. The function CanIf_Transmit is re-entrant. Therefore the function Can_Write needs to be implemented thread-safe (for example by using mutexes):

- Further (preemptive) calls will return with CAN_BUSY when the write can’t be performed re-entrant. (example: write to different hardware TX Handles allowed, write to same TX Handles not allowed)
- In case of CAN_BUSY the CanIf module queues that request. (same behavior as if all hardware objects are busy)
- Can_EnableCanInterrupts and Can_DisableCanInterrupts may be called inside re-entrant functions. Therefore these functions also need to be reentrant.
- All other services don’t need to be implemented as reentrant functions.
- The CAN main functions (i.e. Can_MainFunction_Read) shall not be interrupted by themselves. Therefore these CAN main functions are not reentrant.

7.10 Pretended Networking

Optimizing energy efficiency is becoming increasingly important in all automotive domains since energy consumption has direct impact on fuel consumption, CO2 emissions, and range of hybrid or all electric vehicles. The concept of Pretended Networking has a high potential for energy reduction on ECU level.

(a) ICOM Software implementation: ICOM features are exclusively implemented in software, reusing the existing Communication Controller

(b) ICOM is a functional hardware extension of the existing Communication Controller

(c) ICOM is a separate hardware component with its own Communication Controller

Figure 7-6: Possible ICOM implementations
The use of Intelligent Communication Controllers (ICOM) supports those features (no specific hardware implementation mandatory). If some or all of the functionality of an ECU is temporarily not required, e.g., based on the vehicle state, the ECU can enter a “Pretended Networking” mode. In this mode, the MCU and/or peripherals are switched into a low-power mode. Only the ICOM and the connected transceivers stay active. The ICOM generates a wakeup event, caused by, e.g. a received bus message, when the ECU needs to resume operation. Depending on the ICOM implementation, message ID and payload of received messages may either be evaluated and filtered completely in hardware, require a callback mechanism in software. As shown in Figure 7-6, the ECU implementation can be divided into three possible variants – Approach (a) software approach, without specific hardware to support Pretended Networking. Approach (b) depicts a functional hardware extension of a communication controller and approach (c) shows a hardware variant with a 2nd extended communication controller for wakeup handling. All variants and ICOM implementations shall be supported by Pretended Networking.

Depending on the hardware implementation, the ICOM is also able to send messages. By using the ICOM to continue to send, e.g., status messages, other nodes that rely on that message are not affected by an ECU in Pretended Networking mode.

Furthermore, Pretended Networking aims at reducing wakeup response time, i.e., the time between a wakeup event and valid behavior of an ECU. By using the ICOM to save relevant messages during activated Pretended Networking mode, the application has access to the last valid signal values directly after resuming operation. Therefore, the ECU can immediately respond to a user request after wakeup and does not have to wait until the according message is received again.

7.10.1 Support Pretended Networking mode handling

[SWS_CAN_00497] [ The CAN driver shall deactivate Pretended Networking after initialization of the CAN controller. ] ()

Activation of Pretended Networking:

[SWS_CAN_00462] [ Pretended Networking shall be activated by calling Can_SetIcomConfiguration() with a configuration ID not set to 0. ] ()

[SWS_CAN_00464] [ CanDrv is responsible to perform reconfiguration of the CAN Controller (incl. ICOM) according to the CanIcomConfig parameters for the selected configuration (CanIcomConfigId). ] ()

[SWS_CAN_00467] [ If activation was successful then CanIf_CurrentIcomConfiguration shall be called with the parameter Error set to ICOM_SWITCH_E_OK referring to the corresponding CAN controller with the abstract CanIf ControllerId. If activation was not successful then CanIf_CurrentIcomConfiguration shall be called with the parameter Error set to]
ICOM\_SWITCH\_E\_FAILED referring to the corresponding CAN controller with the abstract CanIf ControllerId. ()

[SWS\_CAN\_00468] If Pretended Networking is activated CanDrv shall call CanIf\_RxIndication() if and only if the received message matches the wakeup conditions of the CanIcomConfig (see CanIcomWakeupCauses). ()

[SWS\_CAN\_00470] If Pretended Networking is activated CanDrv shall reject Can\_Write() requests with return value CAN\_BUSY. ()

[SWS\_CAN\_00498] The CAN driver shall deactivate Pretended Networking before the CAN Controller is started by SetControllerMode(CAN\_CS\_STARTED). ()

**Deactivation of Pretended Networking:**

[SWS\_CAN\_00471] Pretended Networking shall be deactivated (i.e. CanDrv shall behave as when it is configured without Pretended Networking support) by calling Can\_SetIcomConfiguration() with a configuration ID = 0. ()

[SWS\_CAN\_00472] If Pretended Networking is deactivated CanDrv shall process the messages normally as configured in the normal configuration. ()

[SWS\_CAN\_00474] CAN driver shall inform CanIf about a configuration switch by calling CanIf\_CurrentIcomConfiguration referring to the corresponding CAN controller with the abstract CanIf ControllerId. The error parameter is set to ICOM\_SWITCH\_E\_OK if deactivation is successful and to ICOM\_SWITCH\_E\_FAILED otherwise. ()

[SWS\_CAN\_00499] The CAN driver shall deactivate Pretended Networking before the CAN Controller is stopped by SetControllerMode(CAN\_CS\_STOPPED). ()

**7.10.2 Support autonomous sending and receiving of messages**

[SWS\_CAN\_00477] Autonomous sending of messages in Pretended Networking mode shall be supported only if additional ICOM hardware is available. A configuration parameter defines if there is hardware support or not (Refer to CanIcomVariant). ()

[SWS\_CAN\_00478] If the ICOM is implemented in software the controller shall not send messages in Pretended Networking mode. ()

[SWS\_CAN\_00479] CanDriver shall forward all received messages received during Pretended Networking Mode to CanIf. ()
7.11 Error classification

[SWS_Can_00104] The Can module shall be able to detect the following errors and exceptions depending on its configuration (default/production)] (SRS_BSW_00337, SRS_BSW_00385, SRS_BSW_00331)

7.11.1 Development Errors

<table>
<thead>
<tr>
<th>Type or error</th>
<th>Relevance</th>
<th>Related error code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Service called with wrong parameter</td>
<td>Development</td>
<td>CAN_E_PARAM_POINTER  CAN_E_PARAM_HANDLE  CAN_E_PARAM_DATA_LENGTH  CAN_E_PARAM_CONTROLLER</td>
<td>0x01 0x02 0x03 0x04</td>
</tr>
<tr>
<td>API Service used without initialization</td>
<td>Development</td>
<td>CAN_E_UNINIT</td>
<td>0x05</td>
</tr>
<tr>
<td>Invalid transition for the current mode</td>
<td>Development</td>
<td>CAN_E_TRANSITION</td>
<td>0x06</td>
</tr>
<tr>
<td>Parameter Baudrate has an invalid value</td>
<td>Development</td>
<td>CAN_E_PARAM_BAUDRATE</td>
<td>0x07</td>
</tr>
<tr>
<td>Invalid Configuration Id ICOM</td>
<td>Development</td>
<td>CAN_E_ICOM_CONFIG_INVALID</td>
<td>0x08</td>
</tr>
<tr>
<td>Invalid configuration set selection</td>
<td>Development</td>
<td>CAN_E_INIT_FAILED</td>
<td>0x09</td>
</tr>
</tbody>
</table>

7.11.2 Runtime Errors

<table>
<thead>
<tr>
<th>Type or error</th>
<th>Relevance</th>
<th>Related error code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received CAN message is lost</td>
<td>Runtime</td>
<td>CAN_E_DATALOST</td>
<td>0x01</td>
</tr>
</tbody>
</table>

[SWS_Can_00026] The Can module shall indicate errors that are caused by erroneous usage of the Can module API. This covers API parameter checks and call sequence errors. ] (SRS_BSW_00337, SRS_BSW_00323, SRS_SPAL_00157)

[SWS_Can_00091] After return of the DET the Can module’s function that raised the development error shall return immediately.] (SRS_SPAL_12448)

[SWS_Can_00089] The Can module’s environment shall indicate development errors only in the return values of a function of the Can module when DET is switched on and the function provides a return value. The returned value is E_NOT_OK. ] (SRS_BSW_00369, SRS_BSW_00386, SRS_SPAL_12448)

7.11.3 Runtime Errors

[SWS_CAN_XXXXX] Runtime Error Types
### 7.11.4 Transient Faults

**[SWS_CAN_XXXXX] Transient Faults Types**

<table>
<thead>
<tr>
<th>Type of error</th>
<th>Related error code</th>
<th>Value [hex]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7.11.5 Production Errors

The Can module does not call the Diagnostic Event Manager, because there is no production error code defined for the Can module.

### 7.11.6 Return Values

CAN_BUSY is reported via return value of the function Can_Write. The CanIf module reacts according the sequence diagrams specified for the CanIf module. E_NOT_OK is reported via return value in case of a wakeup during transition to sleep mode. Bus-off and Wake-up events are forwarded via notification callback functions.

### 7.12 CAN FD Support

For performance reasons some CAN controllers allow to use a Flexible Data-Rate feature called CAN FD (see "CAN with Flexible Data-Rate" specification). Indicated during the arbitration phase it is possible to switch to a higher baud rate during payload and CRC. This second baud rate has to be configured by extending CanControllerBaudrateConfig with CanControllerFdBaudrateConfig. If a baud rate is active which has a CAN FD configuration (see CanControllerFdBaudrateConfig) the CAN FD feature is enabled for this controller. The specified second baud rate is needed to support reception of CAN FD frames with bit rate switch (BRS). Whether the second baudrate is used for transmission or not depends on configuration parameter CanControllerTxBitRateSwitch (see CanControllerFdBaudrateConfig).

However, there may be cases where conventional CAN 2.0 messages need to be transmitted in networks supporting CAN-FD messages for example to facilitate CAN selective wakeup. In these cases it is necessary to support transmitting interleaved
conventional CAN messages with CAN-FD messages. This can be achieved on frame level by using the two most significant bits of the CanId (see Can_IdType, SWS_Can_00416) passed during Can_Write to indicate which kind of frame shall be used.

CAN FD also supports an extended payload which allows the transmission of up to 64 bytes. This feature also depends on the CAN FD configuration (see CanControllerFdBaudrateConfig). Therefore, if the CAN Controller is in CAN FD mode (valid CanControllerFdBaudrateConfig) and the CAN FD flag is set in CanId passed to Can_Write(), CanDrv supports the transmission of PDUs with a length up to 64 bytes. If there is a request to transmit a CAN FD frame and the CAN Controller is not in CAN FD mode (no CanControllerFdBaudrateConfig) the frame is sent as conventional CAN frame as long as the PDU length <= 8 bytes.
8 API specification

The prefix of the function names may be changed in an implementation with several Can modules as described in SWS_Can_00284.

8.1 Imported types

In this chapter all types included from the following files are listed:
[SWS_Can_00222] |
<table>
<thead>
<tr>
<th>Module</th>
<th>Imported Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can_GeneralTypes</td>
<td>Can_ControllerStateType</td>
</tr>
<tr>
<td></td>
<td>Can_ErrorStateType</td>
</tr>
<tr>
<td></td>
<td>Can_HwHandleType</td>
</tr>
<tr>
<td></td>
<td>Can_HwType</td>
</tr>
<tr>
<td></td>
<td>Can_IdType</td>
</tr>
<tr>
<td></td>
<td>Can_PduType</td>
</tr>
<tr>
<td>ComStack_Types</td>
<td>IcomConfigIdType</td>
</tr>
<tr>
<td></td>
<td>IcomSwitch_ErrorType</td>
</tr>
<tr>
<td></td>
<td>PduIdType</td>
</tr>
<tr>
<td></td>
<td>PduInfoType</td>
</tr>
<tr>
<td>EcuM</td>
<td>EcuM_WakeupSourceType</td>
</tr>
<tr>
<td>Icu</td>
<td>Icu_ChannelType</td>
</tr>
<tr>
<td>Os</td>
<td>CounterType</td>
</tr>
<tr>
<td></td>
<td>StatusType</td>
</tr>
<tr>
<td></td>
<td>TickRefType</td>
</tr>
<tr>
<td>Std_Types</td>
<td>Std_ReturnType</td>
</tr>
<tr>
<td></td>
<td>Std_VersionInfoType</td>
</tr>
</tbody>
</table>
| ()
8.2 Type definitions

[SWS_CAN_00487] The types specified in Can_GeneralTypes shall be declared in Can_GeneralTypes.h.

[SWS_CAN_00439] The content of Can_GeneralTypes.h shall be protected by a CAN_GENERAL_TYPES define.

[SWS_CAN_00440] If different CAN drivers are used, only one instance of this file has to be included in the source tree. For implementation all Can_GeneralTypes.h related types in the documents mentioned before shall be considered.

8.2.1 Can_ConfigType

[SWS_CAN_00413] Name: Can_ConfigType
Type: Structure
Range: Implementation specific.

Description: This is the type of the external data structure containing the overall initialization data for the CAN driver and SFR settings affecting all controllers. Furthermore it contains pointers to controller configuration structures. The contents of the initialization data structure are CAN hardware specific.

8.2.2 Can_PduType

[SWS_CAN_00415] Name: Can_PduType
Type: Structure

Element:
- PduIdType swPduHandle
- uint8 length
- Can_IdType id
- uint8* sdu

Description: This type unites PduId (swPduHandle), SduLength (length), SduData (sdu), and CanId (id) for any CAN L-SDU.

8.2.3 Can_IdType

[SWS_CAN_00416] Name: Can_IdType
Type: uint32

Range:
- Standard32Bit: 0..0x400007FF
- Extended32Bit: 0..0xDFFFFFFF

Description: Represents the Identifier of an L-PDU. The two most significant bits specify the frame type:
- 00 CAN message with Standard CAN ID
- 01 CAN FD frame with Standard CAN ID
- 10 CAN message with Extended CAN ID
8.2.4 Can_HwHandleType

[SWS_Can_00429]

Name: Can_HwHandleType
Type: uint8, uint16
Range: Standard -- 0..0xFF
       Extended -- 0..0xFFFF
Description: Represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects use extended range.

8.2.5 Can_HwType

[SWS_CAN_00496]

Name: Can_HwType
Type: Structure
Element: Can_IdType   CanId   Standard/Extended CAN ID of CAN LPDU
        Can_HwHandleType   Hoh   ID of the corresponding Hardware Object Range
        uint8   ControllerId   ControllerId provided by CanIf clearly identify the corresponding controller
Description: This type defines a data structure which clearly provides an Hardware Object Handle including its corresponding CAN Controller and therefore CanDrv as well as the specific CanId.

8.2.6 Can_ReturnType

[SWS_Can_00039]

Range: CAN_BUSY   0x02  transmit request could not be processed because no transmit object was available
Description: Overlayed return value of Std_ReturnType for CAN driver API Can_Write()
(SRS_BSW_00331)

8.2.7 Can_ErrorStateType

[SWS_Can_91003]

Name: Can_ErrorStateType
Type: Enumeration
Range: CAN_ERRORSTATE_ACTIVE -- The CAN controller takes fully part in communication.
       CAN_ERRORSTATE_PASSIVE -- The CAN controller takes part in communication, but does not send active error frames.
       CAN_ERRORSTATE_BUSOFF -- The CAN controller does not take part in
8.2.8 Can_ControllerStateType

<table>
<thead>
<tr>
<th>Name:</th>
<th>Can_ControllerStateType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Range:</td>
<td>CAN CS UNINIT 0x00 CAN controller state UNINIT. CAN CS STARTED 0x01 CAN controller state STARTED. CAN CS STOPPED 0x02 CAN controller state STOPPED. CAN CS SLEEP 0x03 CAN controller state SLEEP.</td>
</tr>
<tr>
<td>Description:</td>
<td>States that are used by the several ControllerMode functions.</td>
</tr>
</tbody>
</table>

8.3 Function definitions

This is a list of functions provided for upper layer modules.

8.3.1 Services affecting the complete hardware unit

8.3.1.1 Can_Init

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Can_Init</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void Can_Init(</td>
</tr>
<tr>
<td></td>
<td>const Can_ConfigType* Config</td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x00</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Config Pointer to driver configuration.</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>This function initializes the module.</td>
</tr>
</tbody>
</table>

Symbolic names of the available configuration sets are provided by the configuration description of the Can module. See chapter 10 about configuration description.

[SWS_Can_00174] If development error detection for the Can module is enabled: The function Can_Init shall raise the error CAN_E_TRANSITION if the driver is not in state CAN_UNINIT. [SWS_Can_00174]
If development error detection for the Can module is enabled: The function Can_Init shall raise the error CAN_E_TRANSITION if the CAN controllers are not in state UNINIT.

8.3.1.2 Can_GetVersionInfo

If development error detection for the Can module is enabled: The function Can_GetVersionInfo shall raise the error CAN_E_PARAM_POINTER if the parameter versionInfo is a null pointer.

8.3.1.3 Can_DeInit

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 Can_DeInit function has to be sure no CAN controller is in the state STARTED.

[SWS_Can_91011] [If development error detection for the Can module is enabled: The function Can_DeInit shall raise the error CAN_E_TRANSITION if the driver is not in state CAN_READY.]
(SRS_BSW_00369)

[SWS_Can_91012] [If development error detection for the Can module is enabled: The function Can_DeInit shall raise the error CAN_E_TRANSITION if any of the CAN controllers is in state STARTED.]
(SRS_BSW_00369)

8.3.2 Services affecting one single CAN Controller

8.3.2.1 Can_SetBaudrate

[SWS_CAN_00491] [Service name: Can_SetBaudrate
Syntax: Std_ReturnType Can_SetBaudrate(
    uint8 Controller,
    uint16 BaudRateConfigID
)
Service ID[hex]: 0x0f
Sync/Async: Synchronous
Reentrancy: Reentrant for different Controllers. Non reentrant for the same Controller.
Parameters (in):
- Controller: CAN controller, whose baud rate shall be set
- BaudRateConfigID: references a baud rate configuration by ID (see CanControllerBaudRateConfigID)
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.

There might be several baud rate configurations available. The function Can_SetBaudrate can be used to switch between different configurations. Depending on the old and new baud rate configuration only a subset of parameters may be changed during runtime and a re-initialization of the CAN Controller might be avoidable.

If the call of Can_SetBaudrate will cause a re-initialization of the CAN Controller the CAN controller must be in state STOPPED when this function is called (see SWS_Can_00256 and SWS_Can_00260).

The CAN controller is in state STOPPED after (re-)initialization (see SWS_Can_00259).

[SWS_CAN_00492] [If development error detection for the Can module is enabled:
The function Can_SetBaudrate shall raise the error CAN_E_UNINIT and return E_NOT_OK if the driver is not yet initialized. | ()

[SWS_CAN_00493] If development error detection for the Can module is enabled: The function Can_SetBaudrate shall raise the error CAN_E_PARAM_BAUDRATE and return E_NOT_OK if the parameter BaudRateConfigID has an invalid value. | ()

[SWS_CAN_00494] If development error detection for the Can module is enabled the function Can_SetBaudrate shall raise the error CAN_E_PARAM_CONTROLLER and return E_NOT_OK if the parameter Controller is out of range. | ()

[SWS_CAN_00500] If the requested baud rate change can not performed without a re-initialization of the CAN Controller E_NO_OK shall be returned. | ()

8.3.2.2 Can_SetControllerMode

[SWS_Can_00230] | 
| Service name: Can_SetControllerMode |
| Syntax: Std_ReturnType Can_SetControllerMode( uint8 Controller, Can_ControllerStateType Transition ) |
| Service ID[hex]: 0x03 |
| Sync/Async: Asynchronous |
| Reentrancy: Non Reentrant |
| Parameters (in): Controller: CAN controller for which the status shall be changed, Transition: Transition value to request new CAN controller state |
| Parameters (inout): None |
| Parameters (out): None |
| Return value: Std_ReturnType E_OK: request accepted, E_NOT_OK: request not accepted, a development error occurred |
| Description: This function performs software triggered state transitions of the CAN controller State machine. |

| () |

[SWS_Can_00017] | The function Can_SetControllerMode shall perform software triggered state transitions of the CAN controller State machine. See also [SRS_SPAL_12169] (SRS_SPAL_12169, SRS_Can_01053) |

[SWS_Can_00384] | Each time the CAN controller state machine is triggered with the state transition value CAN_CS_STARTED, the function Can_SetControllerMode shall re-initialize the CAN controller with the same controller configuration set previously used by functions Can_SetBaudrate or Can_Init. | ()
Refer to SWS_Can_00048 for the case of a wakeup event from CAN bus occurred during sleep transition. |

[SWS_Can_00294] | The function Can_SetControllerMode shall disable the wake-up interrupt, while checking the wake-up status. | ()
### 8.3.2.3 Can_DisableControllerInterrupts

[SWS_Can_00231]  

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Can_DisableControllerInterrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>void Can_DisableControllerInterrupts(</td>
</tr>
<tr>
<td></td>
<td>uint8 Controller</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0x04</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Controller CAN controller for which interrupts shall be disabled.</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>This function disables all interrupts for this CAN controller.</td>
</tr>
</tbody>
</table>

(SRS_BSW_00312)

[SWS_Can_00049] [The function Can_DisableControllerInterrupts shall access the CAN controller registers to disable all interrupts for that CAN controller only, if interrupts for that CAN Controller are enabled.] (SRS_Can_01043)

[SWS_Can_00202] [When Can_DisableControllerInterrupts has been called several times, Can_EnableControllerInterrupts must be called as many times before the interrupts are re-enabled.] ()

Implementation note:
The function Can_DisableControllerInterrupts can increase a counter on every execution that indicates how many Can_EnableControllerInterrupts need to be called before the interrupts will be enabled (incremental disable).

[SWS_Can_00204] [ The Can module shall track all individual enabling and disabling of interrupts in other functions (i.e. Can_SetControllerMode) , so that the correct interrupt enable state can be restored. ] ()

Implementation example:
- in 'interrupts enabled mode': For each interrupt state change does not only modify the interrupt enable bit, but also a software flag.
- in 'interrupts disabled mode': only the software flag is modified.
- Can_DisableControllerInterrupts and Can_EnableControllerInterrupts do not modify the software flags.
- Can_EnableControllerInterrupts reads the software flags to re-enable the correct interrupts.

[SWS_Can_00205] [ If development error detection for the Can module is enabled: The function Can_DisableControllerInterrupts shall raise the error CAN_E_UNINIT if the driver not yet initialized. ] ()

[SWS_Can_00206] [ If development error detection for the Can module is enabled: The function Can_DisableControllerInterrupts shall raise the error CAN_E_PARAM_CONTROLLER if the parameter Controller is out of range. ] ()

8.3.2.4 Can_EnableControllerInterrupts

[SWS_Can_00232] [ Service name: Can_EnableControllerInterrupts
Syntax: void Can_EnableControllerInterrupts( uint8 Controller )
Service ID[hex]: 0x05
Sync/Async: Synchronous
Reentrancy: Reentrant
Parameters (in): Controller CAN controller for which interrupts shall be re-enabled
Parameters (inout): None
Parameters (out): None
Return value: None
Description: This function enables all allowed interrupts.
](SRS_BSW_00312)

[SWS_Can_00050] [ The function Can_EnableControllerInterrupts shall enable all interrupts that must be enabled according the current software status. ] (SRS_Can_01043)
SWS_Can_00202 applies to this function.

[SWS_Can_00208] [ The function Can_EnableControllerInterrupts shall perform no action when Can_DisableControllerInterrupts has not been called before. ] ()
See also implementation example for Can_DisableControllerInterrupts.
If development error detection for the Can module is enabled: The function Can_EnableControllerInterrupts shall raise the error CAN_E_UNINIT if the driver not yet initialized. ()

If development error detection for the Can module is enabled: The function Can_EnableControllerInterrupts shall raise the error CAN_E_PARAM_CONTROLLER if the parameter Controller is out of range. ()

### 8.3.2.5 Can_CheckWakeup

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Can_CheckWakeup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Std_ReturnType Can_CheckWakeup(</td>
</tr>
<tr>
<td></td>
<td>uint8 Controller</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Controller</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>Std_ReturnType E_OK: API call has been accepted</td>
</tr>
<tr>
<td></td>
<td>E_NOT_OK: API call has not been accepted</td>
</tr>
<tr>
<td>Description:</td>
<td>This function checks if a wakeup has occurred for the given controller.</td>
</tr>
</tbody>
</table>

The function Can_CheckWakeup shall check if the requested CAN controller has detected a wakeup. If a wakeup event was successfully detected, reporting shall be done to EcuM via API EcuM_SetWakeupEvent. ()

The function Can_CheckWakeup shall be pre compile time configurable On/Off by the configuration parameter: CanWakeupFunctionalityAPI ()

If development error detection for the Can module is enabled: The function Can_CheckWakeup shall raise the error CAN_E_UNINIT if the driver is not yet initialized. ()

If development error detection for the Can module is enabled: The function Can_CheckWakeup shall raise the error CAN_E_PARAM_CONTROLLER if the parameter Controller is out of range. ()

### 8.3.2.6 Can_GetControllerErrorState

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Can_GetControllerErrorState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Std_ReturnType Can_GetControllerErrorState(</td>
</tr>
<tr>
<td></td>
<td>uint8 ControllerId,</td>
</tr>
<tr>
<td></td>
<td>Can_ErrorStateType* ErrorStatePtr</td>
</tr>
<tr>
<td>Description:</td>
<td>This function gets the error state of the given controller.</td>
</tr>
</tbody>
</table>

The function Can_GetControllerErrorState shall get the error state of the given controller. ()
### 8.3.2.7 Can_GetControllerMode

**Service name:** Can_GetControllerMode  
**Syntax:**  
```c
Std_ReturnType Can_GetControllerMode(
    uint8 Controller,
    Can_ControllerStateType* ControllerModePtr
)
```

<table>
<thead>
<tr>
<th>Service ID[hex]:</th>
<th>0x12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync/Async:</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Non Reentrant</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Controller: CAN controller for which the status shall be requested.</td>
</tr>
<tr>
<td>Parameters ( inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>ControllerModePtr: Pointer to a memory location, where the current mode of the CAN controller will be stored.</td>
</tr>
<tr>
<td>Return value:</td>
<td>Std_ReturnType: E_OK: Controller mode request has been accepted. E_NOT_OK: Controller mode request has not been accepted.</td>
</tr>
</tbody>
</table>

**Description:** This service obtains the mode of the CAN controller.

---

[SWS_Can_91005] If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can_GetControllerErrorState shall raise development error CAN_E_UNINIT and return E_NOT_OK.] (SRS_BSW_00406, SRS_BSW_00416)

[SWS_Can_91006] If development error detection for the Can module is enabled: if the parameter ControllerId is out of range, the function Can_GetControllerErrorState shall raise development error CAN_E_PARAM_CONTROLLER and return E_NOT_OK.] (SRS_BSW_00323)

[SWS_Can_91007] If development error detection for the Can module is enabled: if the parameter ErrorStatePtr is a null pointer, the function Can_GetControllerErrorState shall raise development error CAN_E_PARAM_POINTER and return E_NOT_OK.] (SRS_BSW_00323)

[SWS_Can_91008] When the API Can_GetControllerErrorState() is called with Controller Id as input parameter then Can driver shall read the error state register of Can Controller and shall return the error status to upper layer.] (SRS_CAN_01167)
<table>
<thead>
<tr>
<th>Service name</th>
<th>Can_Write</th>
</tr>
</thead>
</table>
| Syntax:      |.Std_ReturnType Can_Write(
|              |    Can_HwHandleType Hth,
|              |    const Can_PduType* PduInfo
|              |) |
| Service ID[hex]: | 0x06 |
| Sync/Async:  | Synchronous |
| Reentrancy:  | Reentrant (thread-safe) |
| Parameters (in): | Hth Information which HW-transmit handle shall be used for transmit. 
|                | PduInfo Pointer to SDU user memory, Data Length and Identifier. |
| Parameters (inout): | None |
| Parameters (out): | None |
| Return value: | Std_ReturnType 
|              | E_OK: Write command has been accepted 
|              | E_NOT_OK: development error occurred 
|              | CAN_BUSY: No TX hardware buffer available or pre-emptive call of Can_Write that can't be implemented re-entrant (see |
| Description: | This function is called by CanIf to pass a CAN message to CanDrv for transmission. |

The function Can_Write first checks if the hardware transmit object that is identified by the HTH is free and if another Can_Write is ongoing for the same HTH.

[SWS_Can_00212] The function Can_Write shall perform following actions if the hardware transmit object is free:

- The mutex for that HTH is set to ‘signaled’
- The ID, Data Length and SDU are put in a format appropriate for the hardware (if necessary) and copied in the appropriate hardware registers/buffers.
- All necessary control operations to initiate the transmit are done
- The mutex for that HTH is released
- The function returns with E_OK (SRS_Can_01049)

[SWS_Can_00213] The function Can_Write shall perform no actions if the hardware transmit object is busy with another transmit request for an L-PDU:

1. The transmission of the other L-PDU shall not be cancelled and the function Can_Write is left without any actions.
2. The function Can_Write shall return CAN_BUSY. (SRS_Can_01049).

[SWS_Can_00214] The function Can_Write shall return CAN_BUSY if a preemptive call of Can_Write has been issued, that could not be handled reentrant (i.e. a call with the same HTH). (SRS_BSW_00312, SRS_Can_01049)

[SWS_Can_00275] The function Can_Write shall be non-blocking. ()

[SWS_Can_00216] If development error detection for the Can module is enabled: The function Can_Write shall raise the error CAN_E_UNINIT and shall return E_NOT_OK if the driver is not yet initialized. ()

[SWS_Can_00217] If development error detection for the Can module is enabled: The function Can_Write shall raise the error CAN_E_PARAM_HANDLE and shall return E_NOT_OK if the parameter Hth is not a configured Hardware Transmit Handle. ()

[SWS_Can_00218] The function Can_Write shall return E_NOT_OK and if development error detection for the CAN module is enabled shall raise the error CAN_E_PARAM_DATA_LENGTH:

- If the length is more than 64 byte.
- If the length is more than 8 byte and the CAN controller is not in CAN FD mode (no CanControllerFdBaudrateConfig).
- If the length is more than 8 byte and the CAN controller is in CAN FD mode (valid CanControllerFdBaudrateConfig), but the CAN FD flag in Can_PduType->id is not set (refer Can_IdType). (SRS_Can_01005)

[SWS_CAN_00219] If development error detection for CanDrv is enabled: Can_Write() shall raise CAN_E_PARAM_POINTER and shall return E_NOT_OK if the parameter PduInfo is a null pointer. ()
[SWS_CAN_00503] Can_Write() shall accept a null pointer as SDU (Can_PduType.Can_SduPtrType = NULL) if the trigger transmit API is enabled for this hardware object (CanTriggerTransmitEnable = TRUE).

[SWS_CAN_00504] If the trigger transmit API is enabled for the hardware object, Can_Write() shall interpret a null pointer as SDU (Can_PduType.Can_SduPtrType = NULL) as request for using the trigger transmit interface. If so and the hardware object is free, Can_Write() shall call CanIf_TriggerTransmit() with the maximum size of the message buffer to acquire the PDU's data.

Note: Using the message buffer size allows for late changes of the PDU size, e.g. if a container PDU receives another contained PDU between the call to Can_Write() and the call of CanIf_TriggerTransmit().

[SWS_CAN_00505] If development error detection for CanDrv is enabled: Can_Write() shall raise CAN_E_PARAM_POINTER and shall return E_NOT_OK if the trigger transmit API is disabled for this hardware object (CanTriggerTransmitEnable = FALSE) and the SDU pointer inside PduInfo is a null pointer.

[SWS_CAN_00506] Can_Write() shall return E_NOT_OK if the trigger transmit API (CanIf_TriggerTransmit()) returns E_NOT_OK. (SRS_BSW_00449, SRS_BSW_00357, SRS_BSW_00369, SRS_Can_01130)

[SWS_CAN_00486] The CAN Frame has to be sent according to the two most significant bits of Can_PduType->id. The CAN FD frame bit is only evaluated if CAN Controller is in CAN FD mode (valid CanControllerFdBaudrateConfig).

[SWS_CAN_00502] If PduInfo->SduLength does not match possible DLC values CanDrv shall use the next higher valid DLC for transmission with initialization of unused bytes to the value of the corresponding CanFdPaddingValue (see ECUC_Can_00485). (SRS_Can_01160)

8.4 Call-back notifications

This chapter lists all functions provided by the Can module to lower layer modules. The lower layer module of Can module is the SPI module. The SPI module, which is part of the MCAL, may used to exchange data between the microcontroller and an external CAN controller.

The Can module does not provide callback functions. Only synchronous MCAL API may used to access external CAN controllers.

8.4.1 Call-out function

The AUTOSAR CAN module supports optional L-PDU callouts on every reception of a L-PDU.

[SWS_Can_00443]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>&lt;LPDU_CalloutName&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>boolean &lt;LPDU_CalloutName&gt;(</td>
</tr>
</tbody>
</table>
### 8.4.2 Enabling/Disabling wakeup notification

[SWS_Can_00445] Can driver shall use the following APIs provided by Icu driver, to enable and disable the wakeup event notification:

- Icu_EnableNotification
- Icu_DisableNotification

[SWS_Can_00446] Icu_EnableNotification shall be called when “external” Can controllers have been transitioned to SLEEP state.

[SWS_Can_00447] Icu_DisableNotification shall be called when “external” Can controllers have been transitioned to STOPPED state.

### 8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non-reentrant.

[SWS_Can_00110] There is no requirement regarding the execution order of the CAN main processing functions. (SRS_BSW_00428)

#### 8.5.1.1 Can_MainFunction_Write

[SWS_Can_00225]
8.5.1.2 Can_MainFunction_Write

The function Can_MainFunction_Write shall perform the polling of TX confirmation when CanTxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled.

The function Can_MainFunction_Write() shall obey the following pattern:
- Can_MainFunction_Write_0()
- Can_MainFunction_Write_1()
- Can_MainFunction_Write_2()
- Can_MainFunction_Write_3()
- ... and so on, if more than one period (see ECUC_Can_00356) is supported.

8.5.1.2 Can_MainFunction_Read

The function Can_MainFunction_Read shall perform the polling of RX indications when CanRxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled.

The function Can_MainFunction_Read() shall obey the following pattern:
- Can_MainFunction_Read_0()
- Can_MainFunction_Read_1()
- Can_MainFunction_Read_2()
- Can_MainFunction_Read_3()
• ... and so on, if more than one period (see ECUC_Can_00358) is supported.] ()

8.5.1.3 Can_MainFunction_BusOff

[SWS_Can_00227] |
| Service name: | Can_MainFunction_BusOff |
| Syntax: | void Can_MainFunction_BusOff( void ) |
| Service ID[hex]: | 0x09 |
| Description: | This function performs the polling of bus-off events that are configured statically as 'to be polled'. |

[SWS_Can_00109] | The function Can_MainFunction_BusOff shall perform the polling of bus-off events that are configured statically as 'to be polled'.] ()
(SRS_BSW_00432, SRS_SPAL_00157)

[SWS_Can_00183] | The Can module may implement the function Can_MainFunction_BusOff as empty define in case no polling at all is used.] ()

8.5.1.4 Can_MainFunction_Wakeup

[SWS_Can_00228] |
| Service name: | Can_MainFunction_Wakeup |
| Syntax: | void Can_MainFunction_Wakeup( void ) |
| Service ID[hex]: | 0x0a |
| Description: | This function performs the polling of wake-up events that are configured statically as 'to be polled'. |

[SWS_Can_00112] | The function Can_MainFunction_Wakeup shall perform the polling of wake-up events that are configured statically as 'to be polled'.] ()
(SRS_BSW_00432, SRS_SPAL_00157)

[SWS_Can_00185] | The Can module may implement the function Can_MainFunction_Wakeup as empty define in case no polling at all is used.] ()

8.5.1.5 Can_MainFunction_Mode

[SWS_Can_00368] |
| Service name: | Can_MainFunction_Mode |
| Syntax: | void Can_MainFunction_Mode( void ) |
| Service ID[hex]: | 0x0c |
| Description: | This function performs the polling of CAN controller mode transitions. |

()}
The function Can_MainFunction_Mode shall implement the polling of CAN status register flags to detect transition of CAN Controller state. Compare to chapter 7.3.2.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module. All callback functions that are called by the Can module are implemented in the CanIf module. These callback functions are not configurable.

<table>
<thead>
<tr>
<th>API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanIf_ControllerBusOff</td>
<td>This service indicates a Controller BusOff event referring to the corresponding CAN Controller with the abstract CanIf ControllerId.</td>
</tr>
<tr>
<td>CanIf_ControllerModeIndication</td>
<td>This service indicates a controller state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId.</td>
</tr>
<tr>
<td>CanIf_RxIndication</td>
<td>This service indicates a successful reception of a received CAN Rx L-PDU to the CanIf after passing all filters and validation checks.</td>
</tr>
<tr>
<td>CanIf_TxConfirmation</td>
<td>This service confirms a previously successfully processed transmission of a CAN TxPDU.</td>
</tr>
<tr>
<td>Det_ReportRuntimeError</td>
<td>Service to report runtime errors. If a callout has been configured then this callout shall be called.</td>
</tr>
<tr>
<td>GetCounterValue</td>
<td>This service reads the current count value of a counter (returning either the hardware timer ticks if counter is driven by hardware or the software ticks when user drives counter).</td>
</tr>
</tbody>
</table>

8.6.2 Optional Interfaces

This chapter defines all interfaces that are required to fulfill an optional functionality of the module.

<table>
<thead>
<tr>
<th>API function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanIf_CurrentIcomConfiguration</td>
<td>This service shall inform about the change of the Icom Configuration of a CAN controller using the abstract CanIf ControllerId.</td>
</tr>
<tr>
<td>CanIf_TriggerTransmit</td>
<td>Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr-&gt;SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr-&gt;SduDataPtr and update the length of the actual copied data in PduInfoPtr-&gt;SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.</td>
</tr>
<tr>
<td>Det_ReportError</td>
<td>Service to report development errors.</td>
</tr>
<tr>
<td>EcuM_CheckWakeup</td>
<td>This callout is called by the EcuM to poll a wakeup source. It shall also be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the same interrupt.</td>
</tr>
</tbody>
</table>
8.6.3 Configurable interfaces

There is no configurable target for the Can module. The Can module always reports to CanIf module.

8.7 API supporting Pretended Networking

[SWS_CAN_91001] [ ]

<table>
<thead>
<tr>
<th>Service name:</th>
<th>Can_SetIcomConfiguration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>Std_ReturnType Can_SetIcomConfiguration( uint8 Controller, IcomConfigIdType ConfigurationId )</td>
</tr>
<tr>
<td>Service ID[hex]:</td>
<td>0xf</td>
</tr>
<tr>
<td>Sync/Async:</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Reentrancy:</td>
<td>Reentrant only for different controller Ids</td>
</tr>
<tr>
<td>Parameters (in):</td>
<td>Controller CAN controller for which the status shall be changed. ConfigurationId Requested Configuration</td>
</tr>
<tr>
<td>Parameters (inout):</td>
<td>None</td>
</tr>
<tr>
<td>Parameters (out):</td>
<td>None</td>
</tr>
<tr>
<td>Return value:</td>
<td>Std_ReturnType E_OK: CAN driver succeeded in setting a configuration with a valid Configuration id. E_NOT_OK: CAN driver failed to set a configuration with a valid Configuration id.</td>
</tr>
<tr>
<td>Description:</td>
<td>This service shall change the Icom Configuration of a CAN controller to the requested one.</td>
</tr>
</tbody>
</table>

[SWS_CAN_00480] [ ] The interface Can_SetIcomConfiguration() shall activate or deactivate Pretended Networking and load the requested ICOM configuration for a given controller. ] ()

[SWS_CAN_00481] [ ] If the requested ConfigurationId is not 0, the function Can_SetIcomConfiguration() shall reconfigure the controller with the ICOM configuration parameters of the CanIcomConfig container which CanIcomConfigId matches the requested ConfigurationId. ] ()

[SWS_CAN_00495] [ ] Can_SetIcomConfiguration() shall be pre compile time configurable ON/OFF by the configuration parameter CAN_PUBLIC_ICOM_SUPPORT. ] ()
[SWS_CAN_00475] If development error detection for CanDrv is enabled, then function Can_SetIcomConfiguration() shall report the development error CAN_E_ICOM_CONFIG_INVALID if it is called with an invalid ConfigurationId (i.e. neither 0 nor any of the configured CanIcomConfigId). ] ()
9 Sequence diagrams

9.1 Interaction between Can and CanIf module

For sequence diagrams see the CanIf module Specification [5]. There are described the sequences for Transmission, Reception and Error Handling.

9.2 Wakeup sequence

For Wakeup sequence diagrams refer to Specification of ECU State Manager [7].
10 Configuration specification

This chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the Can module.

Chapter 10.3 specifies published information of the Can module.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8. The described parameters are input for the Can module configurator.

[SWS_Can_00022] The code configuration of the Can module is CAN controller specific. If the CAN controller is sited on-chip, the code generation tool for the Can module is µController specific. If the CAN controller is an external device, the generation tool must not be µController specific.] (SRS_BSW_00159)

[SWS_Can_00024] The valid values that can be configured are hardware dependent. Therefore the rules and constraints can’t be given in the standard. The configuration tool is responsible to do a static configuration checking, also regarding dependencies between modules (i.e. Port driver, MCU driver etc.)] (SRS_BSW_00167, SRS_SPAL_12463)
Figure 10-1: Can Module Configuration Layout
Figure 10-2: Can Controller Configuration Layout
**Figure 10-3: Can Controller Baud Rate Configuration Layout**

- **CanControllerBaudrateConfig**:
  - **EcucParamConfContainerDef**
  - **CanControllerBaudRate**:
    - **EcucIntegerParamDef**
    - min = 0
    - max = 2000

- **CanControllerPropSeg**:
  - **EcucIntegerParamDef**
  - min = 0
  - max = 255

- **CanControllerSeg1**:
  - **EcucIntegerParamDef**
  - min = 0
  - max = 255

- **CanControllerSeg2**:
  - **EcucIntegerParamDef**
  - min = 0
  - max = 255

- **CanControllerSyncJumpWidth**:
  - **EcucIntegerParamDef**
  - min = 0
  - max = 255

- **CanControllerBaudrateConfigID**:
  - **EcucIntegerParamDef**
  - min = 0
  - max = 65535
  - defaultValue = 0

- **CanControllerFdBaudrateConfig**:
  - **EcucParamConfContainerDef**
  - lowerMultiplicity = 0
  - upperMultiplicity = 1
Figure 10-4: Can General Configuration Layout
CanControllerFdBaudrateConfig:
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
+parameter
+parameter
CanControllerFdBaudRate:
EcucIntegerParamDef
min = 0
max = 16000
+parameter
CanControllerTxBitRateSwitch:
EcucBooleanParamDef
defaultValue = true
+parameter
CanControllerPropSeg:
EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerSeg1:
EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerSeg2:
EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerSyncJumpWidth:
EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerTrcvDelayCompensationOffset:
EcucIntegerParamDef
min = 0
max = 400
lowerMultiplicity = 0
upperMultiplicity = 1

Figure 10-5: CanControllerFdBaudrateConfig
**Specification of CAN Driver**

AUTOSAR CP Release 4.3.1

**Figure 10-6: Can Hardware Object Configuration Layout**
CanIcom:
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1

CanIcomConfig:
EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *

CanIcomConfigId:
EcucIntegerParamDef
min = 1
max = 255

CanIcomWakeOnBusOff:
EcucBooleanParamDef
defaultValue = true

CanIcomWakeupCauses:
EcucParamConfContainerDef
+subContainer
+parameter
+parameter
+subContainer

Figure 10-7: CanICOM Layout

CanIcomGeneral:
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1

CanIcomLevel:
EcucEnumerationParamDef
defaultValue = CAN_ICOM_LEVEL_ONE
lowerMultiplicity = 0
upperMultiplicity = 1

CAN_ICOM_LEVEL_ONE:
EcucEnumerationLiteralDef

CAN_ICOM_LEVEL_TWO:
EcucEnumerationLiteralDef

CanIcomVariant:
EcucEnumerationParamDef
defaultValue = CAN_ICOM_VARIANT_NONE
lowerMultiplicity = 0
upperMultiplicity = 1

CAN_ICOM_VARIANT_NONE:
EcucEnumerationLiteralDef

CAN_ICOM_VARIANT_SW:
EcucEnumerationLiteralDef

CAN_ICOM_VARIANT_HW:
EcucEnumerationLiteralDef

Figure 10-8: CanICOM General Configuration Layout
CanIcomRxMessageSignalConfig:
EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *

CanIcomSignalRef:
EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1

CanIcomSignalOperation:
EcucEnumerationParamDef
AND:
EcucEnumerationLiteralDef
XOR:
EcucEnumerationLiteralDef
SMALLER:
EcucEnumerationLiteralDef
EQUAL:
EcucEnumerationLiteralDef
GREATER:
EcucEnumerationLiteralDef

CanIcomSignalValue:
EcucIntegerParamDef
min = 0

CanIcomSignalMask:
EcucIntegerParamDef
min = 0

Figure 10-9: CanIcomRxMessageSignal Configuration Layout
Figure 10-10: CanIcomWakeupCauses Configuration Layout
### 10.2.1 Can

**SWS Item** | ECUC_Can_00489 :
--- | ---
**Module Name** | Can
**Module Description** | This container holds the configuration of a single CAN Driver.
**Post-Build Variant Support** | true
**Supported Config Variants** | VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

#### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanConfigSet</td>
<td>1</td>
<td>This container contains the configuration parameters and sub containers of the AUTOSAR Can module.</td>
</tr>
<tr>
<td>CanGeneral</td>
<td>1</td>
<td>This container contains the parameters related each CAN Driver Unit.</td>
</tr>
</tbody>
</table>

### 10.2.2 CanGeneral

**SWS Item** | ECUC_Can_00328 :
--- | ---
**Container Name** | CanGeneral
**Description** | This container contains the parameters related each CAN Driver Unit.

#### Configuration Parameters

**SWS Item** | ECUC_Can_00064 :
--- | ---
**Name** | CanDevErrorDetect
**Parent Container** | CanGeneral
**Description** | Switches the development error detection and notification on or off.
- true: detection and notification is enabled.
- false: detection and notification is disabled.
**Multiplicity** | 1
**Type** | EcucBooleanParamDef
**Default value** | false
**Post-Build Variant Value** | false
**Value Configuration Class** | Pre-compile time: X | All Variants
| Link time | -- |
| Post-build time | -- |
**Scope / Dependency** | scope: local

**SWS Item** | ECUC_Can_00064 :
--- | ---
**Name** | CanIndex
**Parent Container** | CanGeneral
**Description** | Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0.
**Multiplicity** | 1
**Type** | EcucIntegerParamDef
**Range** | 0 .. 255
**Default value** | --
**Post-Build Variant Value** | false
### Value Configuration Class

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

**Scope / Dependency**
- scope: local

### SWS Item

**ECUC_Can_00434**

**Name**
CanLPduReceiveCalloutFunction

**Parent Container**
CanGeneral

**Description**
This parameter defines the existence and the name of a callout function that is called after a successful reception of a received CAN Rx L-PDU. If this parameter is omitted no callout shall take place.

**Multiplicity**
0..1

**Type**
EcucFunctionNameDef

**Default value**
--

**maxLength**
--

**minLength**
--

**regularExpression**
--

**Post-Build Variant**

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

**Scope / Dependency**
- scope: local

### SWS Item

**ECUC_Can_00355**

**Name**
CanMainFunctionBusoffPeriod

**Parent Container**
CanGeneral

**Description**
This parameter describes the period for cyclic call to Can_MainFunction_Busoff. Unit is seconds.

**Multiplicity**
0..1

**Type**
EcucFloatParamDef

**Range**
0 .. INF

**Default value**
--

**Post-Build Variant**

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

**Scope / Dependency**
- scope: local

### SWS Item

**ECUC_Can_00376**

**Name**
CanMainFunctionModePeriod

**Parent Container**
CanGeneral

**Description**
This parameter describes the period for cyclic call to Can_MainFunction_Mode. Unit is seconds.

**Multiplicity**
1
## Specification of CAN Driver

**AUTOSAR CP Release 4.3.1**

### Type: EcucFloatParamDef

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Default Value</th>
<th>Post-Build</th>
<th>Pre-compile</th>
<th>Link time</th>
<th>Post-build</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>

### Value Configuration Class

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Default Value</th>
<th>Post-Build</th>
<th>Pre-compile</th>
<th>Link time</th>
<th>Post-build</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

### Scope / Dependency

- **SWS Item**: ECUC_Can_00357
  - **Name**: CanMainFunctionWakeupPeriod
  - **Parent Container**: CanGeneral
  - **Description**: This parameter describes the period for cyclic call to Can_MainFunction_Wakeup. Unit is seconds.
  - **Multiplicity**: 0..1

- **Value Configuration Class**
  - **Pre-compile**
    - **X**: All Variants
  - **Link time**
    - **--**
  - **Post-build time**
    - **--**

### Multiplicity Configuration Class

- **Pre-compile**: X
- **Link time**: --
- **Post-build time**: --

### Post-Build Variant Value

- **false**

### SWS Item: ECUC_Can_00095

- **Name**: CanMultiplexedTransmission
- **Parent Container**: CanGeneral
- **Description**: Specifies if multiplexed transmission shall be supported. ON or OFF

- **Type**: EcucBooleanParamDef
- **Default value**
- **Post-Build Variant Value**: false

### Value Configuration Class

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Post-Build</th>
<th>Pre-compile</th>
<th>Link time</th>
<th>Post-build</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td></td>
<td></td>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

### Multiplicity Configuration Class

- **Pre-compile**: X
- **Link time**: --
- **Post-build time**: --

### Scope / Dependency

- **SWS Item**: ECUC_Can_00483
  - **Name**: CanPublicIcomSupport
  - **Parent Container**: CanGeneral
  - **Description**: Selects support of Pretended Network features in Can driver.
    - True: Enabled
    - False: Disabled
  - **Multiplicity**: 1

- **Type**: EcucBooleanParamDef
- **Default value**: false
- **Post-Build Variant Value**: false

### Value Configuration Class

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Post-Build</th>
<th>Pre-compile</th>
<th>Link time</th>
<th>Post-build</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td></td>
<td></td>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

### Scope / Dependency

- **scope**: ECU
- **dependency**: CAN Hardware Unit supports multiplexed transmission

---

- **AUTOSAR confidential**
### Specification of CAN Driver

**AUTOSAR CP Release 4.3.1**

#### Document ID: AUTOSAR_SWS_CANDriver

---

**Scope / Dependency**

<table>
<thead>
<tr>
<th>Post-build time</th>
<th>scope: ECU</th>
</tr>
</thead>
</table>

**SWS Item: ECUC_Can_00482**

**Name**: CanSetBaudrateApi

**Parent Container**: CanGeneral

**Description**: The support of the Can_SetBaudrate API is optional. If this parameter is set to true the Can_SetBaudrate API shall be supported. Otherwise the API is not supported.

**Multiplicity**: 0..1

**Type**: EcucBooleanParamDef

**Default value**: false

**Post-Build Variant**: false

**Post-Build Variant Value**: false

**Multiplicity Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X All Variants</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Value Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X All Variants</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Scope / Dependency**

<table>
<thead>
<tr>
<th>Post-build time</th>
<th>scope: ECU</th>
</tr>
</thead>
</table>

---

**SWS Item: ECUC_Can_00113**

**Name**: CanTimeoutDuration

**Parent Container**: CanGeneral

**Description**: Specifies the maximum time for blocking function until a timeout is detected. Unit is seconds.

**Multiplicity**: 1

**Type**: EcucFloatParamDef

**Range**: [1E-6 .. 65.535]

**Default value**: --

**Post-Build Variant Value**: false

**Value Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X All Variants</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Scope / Dependency**

<table>
<thead>
<tr>
<th>Post-build time</th>
<th>scope: local</th>
</tr>
</thead>
</table>

---

**SWS Item: ECUC_Can_00106**

**Name**: CanVersionInfoApi

**Parent Container**: CanGeneral

**Description**: Switches the Can_GetVersionInfo() API ON or OFF.

**Multiplicity**: 1

**Type**: EcucBooleanParamDef

**Default value**: false

**Post-Build Variant Value**: false

**Value Configuration Class**

<table>
<thead>
<tr>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X All Variants</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Scope / Dependency**

<table>
<thead>
<tr>
<th>Post-build time</th>
<th>scope: local</th>
</tr>
</thead>
</table>

---

**SWS Item: ECUC_Can_00431**

**Name**: CanOsCounterRef

**Parent Container**: CanGeneral

**Description**: This parameter contains a reference to the OsCounter, which is used by
### Specification of CAN Driver

#### AUTOSAR CP Release 4.3.1

**Document ID:** AUTOSAR_SWS_CANDriver

---

**Included Containers**

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanIcomGeneral</td>
<td>0..1</td>
<td>This container contains the general configuration parameters of the ICOM Configuration.</td>
</tr>
<tr>
<td>CanMainFunctionRWPeriods</td>
<td>0..*</td>
<td>This container contains the parameter for configuring the period for cyclic call to Can_MainFunction_Read or Can_MainFunction_Write depending on the referring item.</td>
</tr>
</tbody>
</table>

#### 10.2.3 CanController

**SWS Item** ECUC_Can_00354 :

**Container Name** CanController

**Description** This container contains the configuration parameters of the CAN controller(s).

**Configuration Parameters**

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00314 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanBusoffProcessing</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanController</td>
</tr>
</tbody>
</table>

**Description** Enables / disables API Can_MainFunction_BusOff() for handling busoff events in polling mode.

**Multiplicity** 1

**Type** EcucEnumerationParamDef

**Range**

- INTERRUPT: Interrupt Mode of operation.
- POLLING: Polling Mode of operation.
### SWS Item: ECUC_Can_00315:
- **Name:** CanControllerActivation
- **Parent Container:** CanController
- **Description:** Defines if a CAN controller is used in the configuration.
- **Multiplicity:** 1
- **Type:** EcucBooleanParamDef
- **Default value:** --
- **Post-Build Variant Value:** false

### Value Configuration Class:
- **Pre-compile time:** X All Variants
- **Link time:** --
- **Post-build time:** --

### Scope / Dependency:
- **scope: local**

### SWS Item: ECUC_Can_00382:
- **Name:** CanControllerBaseAddress
- **Parent Container:** CanController
- **Description:** Specifies the CAN controller base address.
- **Multiplicity:** 1
- **Type:** EcuIntegerParamDef
- **Range:** 0 .. 4294967295
- **Default value:** --
- **Post-Build Variant Value:** false

### Value Configuration Class:
- **Pre-compile time:** X All Variants
- **Link time:** --
- **Post-build time:** --

### Scope / Dependency:
- **scope: local**

### SWS Item: ECUC_Can_00316:
- **Name:** CanControllerId
- **Parent Container:** CanController
- **Description:** This parameter provides the controller ID which is unique in a given CAN Driver. The value for this parameter starts with 0 and continue without any gaps.
- **Multiplicity:** 1
- **Type:** EcuIntegerParamDef (Symbolic Name generated for this parameter)
- **Range:** 0 .. 255
- **Default value:** --
- **Post-Build Variant Value:** false

### Value Configuration Class:
- **Pre-compile time:** X All Variants
- **Link time:** --
- **Post-build time:** --

### Scope / Dependency:
- **scope: ECU**

### SWS Item: ECUC_Can_00317:
- **Name:** CanRxProcessing
- **Parent Container:** CanController
- **Description:** Enables / disables API Can_MainFunction_Read() for handling PDU reception.
### SWS Item: ECUC_Can_00318

**Name**: CanTxProcessing  
**Parent Container**: CanController  
**Description**: Enables / disables API Can_MainFunction_Write() for handling PDU transmission events in polling mode.  
**Multiplicity**: 1  
**Type**: EcucEnumerationParamDef  
**Range**  
- **INTERRUPT**: Interrupt Mode of operation.  
- **MIXED**: Mixed Mode of operation  
- **POLLING**: Polling Mode of operation.  
**Post-Build Variant Value**: false  
**Value Configuration Class**  
- **Pre-compile time**: X All Variants  
- **Link time**: --  
- **Post-build time**: --  
**Scope / Dependency**: scope: local

### SWS Item: ECUC_Can_00466

**Name**: CanWakeupFunctionalityAPI  
**Parent Container**: CanController  
**Description**: Adds / removes the service Can_CheckWakeup() from the code. True: Can_CheckWakeup can be used. False: Can_CheckWakeup cannot be used.  
**Multiplicity**: 1  
**Type**: EcucBooleanParamDef  
**Default value**: false  
**Post-Build Variant Value**: false  
**Value Configuration Class**  
- **Pre-compile time**: X All Variants  
- **Link time**: --  
- **Post-build time**: --  
**Scope / Dependency**: scope: local dependency: H/W should support the wakeup functionality to enable this parameter.

### SWS Item: ECUC_Can_00319

**Name**: CanWakeupProcessing  
**Parent Container**: CanController  
**Description**: Enables / disables API Can_MainFunction_Wakeup() for handling wakeup events in polling mode.  
**Multiplicity**: 1
<table>
<thead>
<tr>
<th>Type</th>
<th>EcucEnumerationParamDef</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>INTERRUPT, POLLING</td>
</tr>
<tr>
<td>Post-Build Variant</td>
<td>false</td>
</tr>
<tr>
<td>Value</td>
<td>Pre-compile time, Link time, Post-build time</td>
</tr>
<tr>
<td>Configuration Class</td>
<td>X, --</td>
</tr>
<tr>
<td>Scope / Dependency</td>
<td>scope: local</td>
</tr>
</tbody>
</table>

**SWS Item** ECUC_Can_00330:

- **Name**: CanWakeupSupport
- **Parent Container**: CanController
- **Description**: CAN driver support for wakeup over CAN Bus.
- **Multiplicity**: 1
- **Type**: EcucBooleanParamDef
- **Value Configuration Class**: Pre-compile time, Link time, Post-build time
- **Value Configuration Class Value**: false
- **Value Configuration Class Configuration Class**: X, --
- **Scope / Dependency**: scope: local

**SWS Item** ECUC_Can_00435:

- **Name**: CanControllerDefaultBaudrate
- **Parent Container**: CanController
- **Description**: Reference to baudrate configuration container configured for the Can Controller.
- **Multiplicity**: 1
- **Type**: Reference to [CanControllerBaudrateConfig]
- **Post-Build Variant Value**: true
- **Value Configuration Class**: Pre-compile time, Link time, Post-build time
- **Value Configuration Class Value**: X, --
- **Value Configuration Class Configuration Class**: VARIANT-PRE-COMPIL, --
- **Scope / Dependency**: scope: local

**SWS Item** ECUC_Can_00313:

- **Name**: CanCpuClockRef
- **Parent Container**: CanController
- **Description**: Reference to the CPU clock configuration, which is set in the MCU driver configuration
- **Multiplicity**: 1
- **Type**: Reference to [McuClockReferencePoint]
- **Post-Build Variant Value**: false
- **Value Configuration Class**: Pre-compile time, Link time, Post-build time
- **Value Configuration Class Value**: X, --
- **Value Configuration Class Configuration Class**: All Variants
- **Scope / Dependency**: scope: local

**SWS Item** ECUC_Can_00359:

- **Name**: CanWakeupSourceRef
- **Parent Container**: CanController
- **Description**: This parameter contains a reference to the Wakeup Source for this controller as defined in the ECU State Manager.
Implementation Type: reference to EcuM_WakeupSourceType

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>0..1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Symbolic name reference to [EcuMWakeupSource]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Build Variant</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Scope / Dependency: scope: local

Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanControllerBaudrateConfig</td>
<td>1..*</td>
<td>This container contains bit timing related configuration parameters of the CAN controller(s).</td>
</tr>
<tr>
<td>CanTTController</td>
<td>0..1</td>
<td>CanTTController is specified in the SWS TTCAN and contains the configuration parameters of the TTCAN controller(s) (which are needed in addition to the configuration parameters of the CAN controller(s)). This container is only included and valid if TTCAN is supported by the controller, enabled (see CanSupportTTCANRef, ECUC_Can_00430), and used.</td>
</tr>
</tbody>
</table>

10.2.4 CanControllerBaudrateConfig

SWS Item: ECUC_Can_00387:

<table>
<thead>
<tr>
<th>Container Name</th>
<th>CanControllerBaudrateConfig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This container contains bit timing related configuration parameters of the CAN controller(s).</td>
</tr>
</tbody>
</table>

Configuration Parameters

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00005:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanControllerBaudRate</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanControllerBaudrateConfig</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the baudrate of the controller in kbps.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>EcuUIntegerParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>0 .. 2000</td>
</tr>
</tbody>
</table>

| Post-Build Variant Value | true |

<table>
<thead>
<tr>
<th>Value Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>X</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Scope / Dependency: scope: local

SWS Item: ECUC_Can_00471:

<table>
<thead>
<tr>
<th>Container Name</th>
<th>CanControllerBaudRateConfigID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Uniquely identifies a specific baud rate configuration. This ID is used by SetBaudrate API.</td>
</tr>
</tbody>
</table>
### Specification of CAN Driver

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00073</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanControllerPropSeg</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanControllerBaudrateConfig</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies propagation delay in time quantas.</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucIntegerParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>0 .. 255</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>true</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time: X VARIANT-PRE-COMPILE, Link time: --, Post-build time: X VARIANT-POST-BUILD</td>
</tr>
<tr>
<td>Scope / Dependency</td>
<td>scope: local</td>
</tr>
</tbody>
</table>

### SWS Item

<table>
<thead>
<tr>
<th>ECUC_Can_00074</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Parent Container</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Multiplicity</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Range</td>
</tr>
<tr>
<td>Default value</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
</tr>
<tr>
<td>Value Configuration Class</td>
</tr>
<tr>
<td>Scope / Dependency</td>
</tr>
</tbody>
</table>

### SWS Item

<table>
<thead>
<tr>
<th>ECUC_Can_00075</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Parent Container</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Multiplicity</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Range</td>
</tr>
<tr>
<td>Default value</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
</tr>
<tr>
<td>Value Configuration Class</td>
</tr>
<tr>
<td>Scope / Dependency</td>
</tr>
</tbody>
</table>

### SWS Item

| ECUC_Can_00383 |
Name | CanControllerSyncJumpWidth  
---|---  
**Parent Container** | CanControllerBaudrateConfig  
**Description** | Specifies the synchronization jump width for the controller in time quantas.  
**Multiplicity** | 1  
**Type** | EcucIntegerParamDef  
**Range** | 0 .. 255  
**Default value** | --  
**Post-Build Variant Value** | true  
**Value Configuration Class** |  
Pre-compile time | X VARIANT-PRE-COMPILE  
Link time | --  
Post-build time | X VARIANT-POST-BUILD  
**Scope / Dependency** | scope: local

**Included Containers**

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th><strong>Scope / Dependency</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>CanControllerFdBaudrateConfig</td>
<td>0..1</td>
<td>This optional container contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN FD frame. If this container exists the controller supports CAN FD frames.</td>
</tr>
</tbody>
</table>

### 10.2.5 CanControllerFdBaudrateConfig

**SWS Item** | ECUC_Can_00473 :  
**Container Name** | CanControllerFdBaudrateConfig  
**Description** | This optional container contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN FD frame. If this container exists the controller supports CAN FD frames.  
**Configuration Parameters**

**SWS Item** | ECUC_Can_00481 :  
**Name** | CanControllerFdBaudRate  
**Parent Container** | CanControllerFdBaudrateConfig  
**Description** | Specifies the data segment baud rate of the controller in kbps.  
**Multiplicity** | 1  
**Type** | EcucIntegerParamDef  
**Range** | 0 .. 16000  
**Default value** | --  
**Post-Build Variant Value** | true  
**Value Configuration Class** |  
Pre-compile time | X VARIANT-PRE-COMPILE  
Link time | --  
Post-build time | X VARIANT-POST-BUILD  
**Scope / Dependency** | scope: local

**SWS Item** | ECUC_Can_00476 :  
**Name** | CanControllerPropSeg  
**Parent Container** | CanControllerFdBaudrateConfig  
**Description** | Specifies propagation delay in time quantas.  
**Multiplicity** | 1  
**Type** | EcucIntegerParamDef  
**Range** | 0 .. 255  
**Default value** | --  
**Post-Build Variant Value** | true
### Value Configuration Class

<table>
<thead>
<tr>
<th>Description</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X VARIANT-PRE-COMPILE</td>
<td>--</td>
<td>X VARIANT-POST-BUILD</td>
<td></td>
</tr>
</tbody>
</table>

### Scope / Dependency

scope: local

---

### SWS Item

**ECUC_Can_00477**

**Name**
CanControllerSeg1

**Parent Container**
CanControllerFdBaudrateConfig

**Description**
Specifies phase segment 1 in time quantas.

**Multiplicity**
1

**Type**
EcucIntegerParamDef

**Range**
0 .. 255

**Default value**
--

**Post-Build Variant Value**
true

---

### Value Configuration Class

<table>
<thead>
<tr>
<th>Description</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X VARIANT-PRE-COMPILE</td>
<td>--</td>
<td>X VARIANT-POST-BUILD</td>
<td></td>
</tr>
</tbody>
</table>

### Scope / Dependency

scope: local

---

### SWS Item

**ECUC_Can_00478**

**Name**
CanControllerSeg2

**Parent Container**
CanControllerFdBaudrateConfig

**Description**
Specifies phase segment 2 in time quantas.

**Multiplicity**
1

**Type**
EcucIntegerParamDef

**Range**
0 .. 255

**Default value**
--

**Post-Build Variant Value**
true

---

### Value Configuration Class

<table>
<thead>
<tr>
<th>Description</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X VARIANT-PRE-COMPILE</td>
<td>--</td>
<td>X VARIANT-POST-BUILD</td>
<td></td>
</tr>
</tbody>
</table>

### Scope / Dependency

scope: local

---

### SWS Item

**ECUC_Can_00479**

**Name**
CanControllerSyncJumpWidth

**Parent Container**
CanControllerFdBaudrateConfig

**Description**
Specifies the synchronization jump width for the controller in time quantas.

**Multiplicity**
1

**Type**
EcucIntegerParamDef

**Range**
0 .. 255

**Default value**
--

**Post-Build Variant Value**
true

---

### Value Configuration Class

<table>
<thead>
<tr>
<th>Description</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>X VARIANT-PRE-COMPILE</td>
<td>--</td>
<td>X VARIANT-POST-BUILD</td>
<td></td>
</tr>
</tbody>
</table>

### Scope / Dependency

scope: local

---

### SWS Item

**ECUC_Can_00480**

**Name**
CanControllerTrcvDelayCompensationOffset

**Parent Container**
CanControllerFdBaudrateConfig

**Description**
Specifies the Transceiver Delay Compensation Offset in ns. If not specified Transceiver Delay Compensation is disabled.

**Multiplicity**
0..1

**Type**
EcucIntegerParamDef

---

---

- AUTOSAR confidential -
### Specification of CAN Driver

**AUTOSAR CP Release 4.3.1**

**Document ID**: AUTOSAR_SWS_CANDriver

---

**Range**: 0 .. 400

**Default value**: --

**Post-Build Variant Multiplicity**: true

**Post-Build Variant Value**: true

<table>
<thead>
<tr>
<th>Multiplicity Configuration Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value Configuration Class</strong></td>
<td>Pre-compile time</td>
<td>Link time</td>
<td>Post-build time</td>
</tr>
<tr>
<td><strong>Scope / Dependency</strong></td>
<td>scope: local</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SWS Item</strong></th>
<th>ECUC_Can_00475 :</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>CanControllerTxBitRateSwitch</td>
</tr>
<tr>
<td><strong>Parent Container</strong></td>
<td>CanControllerFdBaudrateConfig</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Specifies if the bit rate switching shall be used for transmissions. If FALSE: CAN FD frames shall be sent without bit rate switching.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcucBooleanParamDef</td>
</tr>
<tr>
<td><strong>Default value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Post-Build Variant Value</strong></td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Value Configuration Class</strong></th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope / Dependency</strong></td>
<td>scope: local</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**No Included Containers**

---

### 10.2.6 CanHardwareObject

<table>
<thead>
<tr>
<th><strong>SWS Item</strong></th>
<th>ECUC_Can_00324 :</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Container Name</strong></td>
<td>CanHardwareObject</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This container contains the configuration (parameters) of CAN Hardware Objects.</td>
</tr>
</tbody>
</table>

#### Configuration Parameters

<table>
<thead>
<tr>
<th><strong>SWS Item</strong></th>
<th>ECUC_Can_00485 :</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>CanFdPaddingValue</td>
</tr>
<tr>
<td><strong>Parent Container</strong></td>
<td>CanHardwareObject</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the value which is used to pad unspecified data in CAN FD frames &gt; 8 bytes for transmission. This is necessary due to the discrete possible values of the DLC if &gt; 8 bytes. If the length of a PDU which was requested to be sent does not match the allowed DLC values, the remaining bytes up to the next possible value shall be padded with this value.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>0 .. 1</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>EcuIntegerParamDef</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>0 .. 255</td>
</tr>
<tr>
<td><strong>Default value</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Post-Build Variant Multiplicity</strong></td>
<td>true</td>
</tr>
</tbody>
</table>
### Post-Build Variant Value

<table>
<thead>
<tr>
<th>Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity Configuration</td>
<td>X</td>
<td>VARIANT-PRE-COMPILE</td>
<td>--</td>
</tr>
</tbody>
</table>

### Value Configuration Class

<table>
<thead>
<tr>
<th>Class</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity Configuration</td>
<td>X</td>
<td>VARIANT-PRE-COMPILE</td>
<td>--</td>
</tr>
</tbody>
</table>

### Scope / Dependency

- **scope: ECU**

### SWS Item: ECUC_Can_00323

- **Name**: CanHandleType
- **Parent Container**: CanHardwareObject
- **Description**: Specifies the type (Full-CAN or Basic-CAN) of a hardware object.
- **Multiplicity**: 1
- **Type**: EcucEnumerationParamDef
- **Range**: BASIC, FULL
  - BASIC: For several L-PDUs are handled by the hardware object.
  - FULL: For only one L-PDU (identifier) is handled by the hardware object.

### SWS Item: ECUC_Can_00490

- **Name**: CanHardwareObjectUsesPolling
- **Parent Container**: CanHardwareObject
- **Description**: Enables polling of this hardware object.
- **Multiplicity**: 0..1
- **Type**: EcucBooleanParamDef
- **Default value**: false

### SWS Item: ECUC_Can_00467

- **Name**: CanHwObjectCount
- **Parent Container**: CanHardwareObject
- **Description**: Number of hardware objects used to implement one HOH. In case of a HRH this parameter defines the number of elements in the hardware FIFO or the number of shadow buffers, in case of a HTH it defines the number of hardware objects used for multiplexed transmission or for a hardware FIFO used by a FullCAN HTH.
- **Multiplicity**: 1
- **Type**: EcuIntegerParamDef
- **Range**: 1 .. 65535
- **Default value**: 1
- **Post-Build Variant Value**: true

### Scope / Dependency

- **dependency**: This parameter shall exist if CanRxProcessing/CanTxProcessing is set to Mixed.
| Class | Link time | -- | | | | Post-build time | X | VARIANT-POST-BUILD | | | | Value Configuration Class | Pre-compile time | X | VARIANT-PRE-COMPILE | | | | Link time | -- | | | | | Post-build time | X | VARIANT-POST-BUILD | | | | Scope / Dependency | scope: ECU | | | | | | | | | | | | | | | | SWS Item | ECUC_Can_00065 | Name | CanIdType | | | Parent Container | CanHardwareObject | | | | | Description | Specifies whether the IdValue is of type | | | | | | | | - standard identifier | | | | | | | - extended identifier | | | | | | | - mixed mode | | | | | | | ImplementationType: Can_IdType | | | | | | | Multiplicity | 1 | | | | | Type | EcucEnumerationParamDef | | | | | | | Range | EXTENDED | All the CANIDs are of type extended only (29 bit). | | | | MIXED | The type of CANIDs can be both Standard or Extended. | | | | STANDARD | All the CANIDs are of type standard only (11 bit). | | | | | Post-Build Variant Value | true | | | | | Value Configuration Class | Pre-compile time | X | VARIANT-PRE-COMPILE | | | | Link time | -- | | | | | Post-build time | X | VARIANT-POST-BUILD | | | | Scope / Dependency | scope: ECU | | | | | | | | | | | | | | | | SWS Item | ECUC_Can_00326 | Name | CanObjectId | | | Parent Container | CanHardwareObject | | | | | Description | Holds the handle ID of HRH or HTH. The value of this parameter is unique in a given CAN Driver, and it should start with 0 and continue without any gaps. The HRH and HTH Ids share a common ID range. Example: HRH0-0, HRH1-1, HTH0-2, HTH1-3 | | | | | | | Multiplicity | 1 | | | | | Type | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | | | | | | Range | 0 .. 65535 | | | | | Default value | -- | | | | | Post-Build Variant Value | false | | | | | Value Configuration Class | Pre-compile time | X | All Variants | | | | Link time | -- | | | | | Post-build time | -- | | | | | Scope / Dependency | scope: ECU | | | | | | | | | | | | | | | | SWS Item | ECUC_Can_00327 | Name | CanObjectType | | | Parent Container | CanHardwareObject | | | | | Description | | | | | | | ImplementationType: CanObjectType | | | | | | | Multiplicity | 1 | | | | | Type | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | | | | | | Range | 0 .. 65535 | | | | | Default value | -- | | | | | Post-Build Variant Value | false | | | | | Value Configuration Class | Pre-compile time | X | All Variants | | | | Link time | -- | | | | | Post-build time | -- | | | | | Scope / Dependency | scope: ECU | | | | | | | | | | | | | | | |
**Description**
Specifies if the HardwareObject is used as Transmit or as Receive object

**Multiplicity**
1

**Type**
EcucEnumerationParamDef

**Range**
| RECEPTIVE | Receive HOH |
| Transmit  | Transmit HOH |

**Post-Build Variant Value**
true

**Value Configuration Class**
- **Pre-compile time**: VARIANT-PRE-COMPILE
- **Post-build time**: VARIANT-POST-BUILD

**SWS Item**
ECUC_Can_00468:

**Name**
CanTriggerTransmitEnable

**Parent Container**
CanHardwareObject

**Description**
This parameter defines if or if not Can supports the trigger-transmit API for this handle.

**Multiplicity**
0..1

**Type**
EcucBooleanParamDef

**Default value**
false

**Value Configuration Class**
- **Pre-compile time**: All Variants
- **Post-build time**:?

**Scope / Dependency**
scope: ECU

**SWS Item**
ECUC_Can_00322:

**Name**
CanControllerRef

**Parent Container**
CanHardwareObject

**Description**
Reference to CAN Controller to which the HOH is associated to.

**Multiplicity**
1

**Type**
Reference to [CanController]

**Post-Build Variant Value**
true

**Value Configuration Class**
- **Pre-compile time**: VARIANT-PRE-COMPILE
- **Post-build time**: VARIANT-POST-BUILD

**Scope / Dependency**
scope: local

**SWS Item**
ECUC_Can_00438:

**Name**
CanMainFunctionRWPeriodRef

**Parent Container**
CanHardwareObject

**Description**
Reference to CanMainFunctionPeriod

**Multiplicity**
0..1

**Type**
Reference to [CanMainFunctionRWPeriods]

**Post-Build Variant Value**
true

**Value Configuration Class**
- **Pre-compile time**: VARIANT-PRE-COMPILE
- **Post-build time**: VARIANT-POST-BUILD

**Scope / Dependency**
scope: local
### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanHwFilter</td>
<td>0..*</td>
<td>This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter.</td>
</tr>
<tr>
<td>CanTTHardwareObjectTrigger</td>
<td>0..*</td>
<td>CanTTHardwareObjectTrigger is specified in the SWS TTCAN and contains the configuration (parameters) of TTCAN triggers for Hardware Objects, which are additional to the configuration (parameters) of CAN Hardware Objects. This container is only included and valid if TTCAN is supported by the controller and, enabled (see CanSupportTTCANRef, ECUC_Can_00430), and used.</td>
</tr>
</tbody>
</table>

### 10.2.7 CanHwFilter

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00468 :</th>
<th>Container Name</th>
<th>CanHwFilter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Configuration Parameters**

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00469 :</th>
<th>Name</th>
<th>CanHwFilterCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td>Parent Container</td>
<td>CanHwFilter</td>
</tr>
<tr>
<td>Specifies (together with the filter mask) the identifiers range that passes the hardware filter.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>EcuIntegerParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>0..4294967295</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>true</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time</td>
</tr>
<tr>
<td></td>
<td>Link time</td>
</tr>
<tr>
<td></td>
<td>Post-build time</td>
</tr>
</tbody>
</table>

### CanHwFilterMask

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00470 :</th>
<th>Name</th>
<th>CanHwFilterMask</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td>Parent Container</td>
<td>CanHwFilter</td>
</tr>
<tr>
<td>Describes a mask for hardware-based filtering of CAN identifiers. The CAN identifiers of incoming messages are masked with the appropriate CanFilterMaskValue. Bits holding a 0 mean don't care, i.e. do not compare the message's identifier in the respective bit position. The mask shall be build by filling with leading 0. In case of CanIdType EXTENDED or MIXED a 29 bit mask shall be build. In case of CanIdType STANDARD a 11 bit mask shall be build.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>EcuIntegerParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>0..4294967295</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>true</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time</td>
</tr>
<tr>
<td></td>
<td>Link time</td>
</tr>
</tbody>
</table>
## 10.2.8 CanConfigSet

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00343</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>CanConfigSet</td>
</tr>
<tr>
<td>Description</td>
<td>This container contains the configuration parameters and sub containers of the AUTOSAR Can module.</td>
</tr>
</tbody>
</table>

### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanController</td>
<td>1..*</td>
<td>This container contains the configuration parameters of the CAN controller(s).</td>
</tr>
<tr>
<td>CanHardwareObject</td>
<td>1..*</td>
<td>This container contains the configuration (parameters) of CAN Hardware Objects.</td>
</tr>
<tr>
<td>CanIcom</td>
<td>0..1</td>
<td>This container contains the parameters for configuring pretended networking</td>
</tr>
</tbody>
</table>

## 10.2.9 CanMainFunctionRWPeriods

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00437</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>CanMainFunctionRWPeriods</td>
</tr>
<tr>
<td>Description</td>
<td>This container contains the parameter for configuring the period for cyclic call to Can_MainFunction_Read or Can_MainFunction_Write depending on the referring item.</td>
</tr>
</tbody>
</table>

### Configuration Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>CanMainFunctionPeriod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Container</td>
<td>CanMainFunctionRWPeriods</td>
</tr>
<tr>
<td>Description</td>
<td>This parameter describes the period for cyclic call to Can_MainFunction_Read or Can_MainFunction_Write depending on the referring item. Unit is seconds. Different poll-cycles will be configurable if more than one CanMainFunctionPeriod is configured. In this case multiple Can_MainFunction_Read() or Can_MainFunction_Write() will be provided by the CAN Driver module.</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucFloatParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>]0 .. INF[</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>false</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scope / Dependency

scope: local
No Included Containers

10.2.10 Canlcom

SWS Item | ECUC_Can_00440 :
--- | ---
Container Name | Canlcom
Description | This container contains the parameters for configuring pretended networking

Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
</table>
| CanlcomConfig  | 1..*         | This container contains the configuration parameters of the ICOM Configuration.

10.2.11 CanlcomConfig

SWS Item | ECUC_Can_00459 :
--- | ---
Container Name | CanlcomConfig
Description | This container contains the configuration parameters of the ICOM Configuration.

Configuration Parameters

SWS Item | ECUC_Can_00441 :
--- | ---
Name | CanlcomConfigId
Parent Container | CanlcomConfig
Description | This parameter identifies the ID of the ICOM configuration.
Multiplicity | 1
Type | EcuIntegerParamDef
Range | 1..255
Default value | --
Post-Build Variant Value | false
Value Configuration Class | Pre-compile time | X | All Variants
| Link time | --
| Post-build time | --
Scope / Dependency | scope: ECU

SWS Item | ECUC_Can_00442 :
--- | ---
Name | CanlcomWakeOnBusOff
Parent Container | CanlcomConfig
Description | This parameter defines that the MCU shall wake if the bus off is detected or not.
Multiplicity | 1
Type | EcuBooleanParamDef
Default value | true
Post-Build Variant Value | false
Value Configuration Class | Pre-compile time | X | All Variants
| Link time | --
| Post-build time | --
Specification of CAN Driver
AUTOSAR CP Release 4.3.1

Scope / Dependency
scope: ECU

Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanIcomWakeupCauses</td>
<td>1</td>
<td>This container contains the configuration parameters of the wake up causes to leave the power saving mode.</td>
</tr>
</tbody>
</table>

10.2.12 CanIcomGeneral

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00444 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Name</td>
<td>CanIcomGeneral</td>
</tr>
<tr>
<td>Description</td>
<td>This container contains the general configuration parameters of the ICOM Configuration.</td>
</tr>
</tbody>
</table>

Configuration Parameters

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00445 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanIcomLevel</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanIcomGeneral</td>
</tr>
<tr>
<td>Description</td>
<td>Defines the level of Pretended Networking. This parameter is reserved for future implementations (Pretended Networking level 2).</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>0..1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>CAN_ICOM_LEVEL_ONE : --</td>
</tr>
<tr>
<td></td>
<td>CAN_ICOM_LEVEL_TWO : --</td>
</tr>
<tr>
<td>Default value</td>
<td>CAN_ICOM_LEVEL_ONE</td>
</tr>
<tr>
<td>Post-Build Variant Multiplicity</td>
<td>false</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>false</td>
</tr>
<tr>
<td>Multiplicity Class Pre-compile time</td>
<td>X All Variants</td>
</tr>
<tr>
<td>Class Link time</td>
<td>--</td>
</tr>
<tr>
<td>Value Pre-build time</td>
<td>X All Variants</td>
</tr>
<tr>
<td>Class Link time</td>
<td>--</td>
</tr>
<tr>
<td>Scope / Dependency</td>
<td>scope: ECU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00446 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanIcomVariant</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanIcomGeneral</td>
</tr>
<tr>
<td>Description</td>
<td>Defines the variant, which is supported by this CanController</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>CAN_ICOM_VARIANT_HW : --</td>
</tr>
<tr>
<td></td>
<td>CAN_ICOM_VARIANT_NONE : --</td>
</tr>
<tr>
<td></td>
<td>CAN_ICOM_VARIANT_SW : --</td>
</tr>
<tr>
<td>Default value</td>
<td>CAN_ICOM_VARIANT_NONE</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>false</td>
</tr>
<tr>
<td>Value Pre-compile time</td>
<td>X All Variants</td>
</tr>
</tbody>
</table>
### 10.2.13 CanIcomRxMessage

**SWS Item** ECUC_Can_00447:

**Container Name** CanIcomRxMessage

**Description** This container contains the configuration parameters for the wakeup causes for matching received messages. It has to be configured as often as received messages are defined as wakeup cause.

**Configuration Parameters**

**SWS Item** ECUC_Can_00448:

**Name** CanIcomCounterValue

**Parent Container** CanIcomRxMessage

**Description** This parameter defines that the MCU shall wake if the message with the ID is received n times on the communication channel.

**Multiplicity** 0..1

**Type** EcucIntegerParamDef

**Range** 1 .. 65536

**Default value** --

**Post-Build Variant** false

**Post-Build Variant Value** false

**Multiplicity Configuration Class** Pre-compile time X All Variants

**Value Configuration Class** Pre-compile time X All Variants

**Scope / Dependency** scope: ECU

**SWS Item** ECUC_Can_00449:

**Name** CanIcomMessageId

**Parent Container** CanIcomRxMessage

**Description** This parameter defines the message ID the wakeup causes of this CanIcomRxMessage are configured for. In addition a mask (CanIcomMessageIdMask) can be defined, in that case it is possible to define a range of rx messages, which can create a wakeup condition.

**Multiplicity** 1

**Type** EcucIntegerParamDef

**Range** 0 .. 536870912

**Default value** --

**Post-Build Variant Value** false
Value Configuration Class | Pre-compile time | X | All Variants  
|--------------------------|----------------|---|--------------  
| Link time                | --             |   |               
| Post-build time          | --             |   |              

Scope / Dependency: scope: ECU

SWS Item: ECUC_Can_00465:
Name: CanIcomMessageIdMask
Parent Container: CanIcomRxMessage
Description: Describes a mask for filtering of CAN identifiers. The CAN identifiers of incoming messages are masked with this CanIcomMessageIdMask. If the masked identifier matches the masked value of CanIcomMessageId, it can create a wakeup condition for this CanIcomRxMessage. Bits holding a 0 mean don't care, i.e. do not compare the message's identifier in the respective bit position. The mask shall be build by filling with leading 0.

Multiplicity: 0..1
Type: EcuIntegerParamDef
Range: 0 .. 536870912
Default value: --
Post-Build Variant Multiplicity: false
Post-Build Variant Value: false

Multiplicity Configuration Class | Pre-compile time | X | All Variants  
|--------------------------|----------------|---|--------------  
| Link time                | --             |   |               
| Post-build time          | --             |   |              

Value Configuration Class | Pre-compile time | X | All Variants  
|--------------------------|----------------|---|--------------  
| Link time                | --             |   |               
| Post-build time          | --             |   |              

Scope / Dependency: scope: ECU
dependency: CanIcomMessageIdMask and CanIcomRxMessageSignalConfig shall not be defined together.

SWS Item: ECUC_Can_00450:
Name: CanIcomMissingMessageTimerValue
Parent Container: CanIcomRxMessage
Description: This parameter defines that the MCU shall wake if the message with the ID is not received for a specific time in s on the communication channel.

Multiplicity: 0..1
Type: EcuFloatParamDef
Range: [INF .. INF]
Default value: --
Post-Build Variant Multiplicity: false
Post-Build Variant Value: false

Multiplicity Configuration Class | Pre-compile time | X | All Variants  
|--------------------------|----------------|---|--------------  
| Link time                | --             |   |               
| Post-build time          | --             |   |              

Value Configuration Class | Pre-compile time | X | All Variants  
|--------------------------|----------------|---|--------------  
| Link time                | --             |   |               
| Post-build time          | --             |   |              

Scope / Dependency: scope: ECU

SWS Item: ECUC_Can_00451:
Name: CanIcomPayloadLengthError
Parent Container: CanIcomRxMessage
This parameter defines that the MCU shall wake if a payload error occurs.

**Description**

**Multiplicity**

1

**Type**

EcucBooleanParamDef

**Default value**

false

**Post-Build Variant Value**

false

**Value Configuration Class**

Pre-compile time: X, All Variants

Link time: --

Post-build time: --

**Scope / Dependency**

scope: ECU

This container contains the configuration parameters for the wakeup causes for matching signals. It has to be configured as often as a signal is defined as wakeup cause. If at least one Signal conditions defined in a CanIcomRxMessageSignalConfig evaluates to true or if no CanIcomRxMessageSignalConfig are defined, the whole wakeup condition is considered to be true. All instances of this container refer to the same frame/pdu (see CanIcomMessageId).

### 10.2.14 CanIcomRxMessageSignalConfig

**SWS Item**

ECUC_Can_00452 :

**Container Name**

CanIcomRxMessageSignalConfig

This container contains the configuration parameters for the wakeup causes for matching signals. It has to be configured as often as a signal is defined as wakeup cause. If at least one Signal conditions defined in a CanIcomRxMessageSignalConfig evaluates to true or if no CanIcomRxMessageSignalConfig are defined, the whole wakeup condition is considered to be true. All instances of this container refer to the same frame/pdu (see CanIcomMessageId).

**Configuration Parameters**

**SWS Item**

ECUC_Can_00487 :

**Name**

CanIcomSignalMask

**Parent Container**

CanIcomRxMessageSignalConfig

This parameter shall be used to mask a signal in the payload of a CAN message. The mask is binary AND with the signal payload. The result will be used in combination of the operations defined in CanIcomSignalOperation with the CanIcomSignalValue.

**Multiplicity**

1

**Type**

EcucIntegerParamDef

**Range**

0 .. 18446744073709551615

**Default value**

--

**Post-Build Variant Value**

false

**Value Configuration Class**

Pre-compile time: X, All Variants

Link time: --
<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00462 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanIcomSignalOperation</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanIcomRxMessageSignalConfig</td>
</tr>
<tr>
<td>Description</td>
<td>This parameter defines the operation, which shall be used to verify the signal value creates a wakeup condition.</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucEnumerationParamDef</td>
</tr>
</tbody>
</table>
| Range           | AND: The received signal value masked by CanIcomSignalMask has at least one bit set in common with CanIcomSignalValue (binary AND).  

EQUA: The received signal value masked by CanIcomSignalMask is equal to CanIcomSignalValue.  

GREA: The received signal value masked by CanIcomSignalMask is strictly greater than CanIcomSignalValue. Values are interpreted as unsigned integers.  

SMAL: The received signal value masked by CanIcomSignalMask is strictly smaller than CanIcomSignalValue. Values are interpreted as unsigned integers.  

XOR: The received signal value masked by CanIcomSignalMask then XORed to CanIcomSignalValue is not null. |
| Post-Build Variant Value | false |
| Value Configuration Class | Pre-compile time: X All Variants |  
| Link time | -- |  
| Post-build time | -- |  
| Scope / Dependency | scope: local |  

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00488 :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>CanIcomSignalValue</td>
</tr>
<tr>
<td>Parent Container</td>
<td>CanIcomRxMessageSignalConfig</td>
</tr>
<tr>
<td>Description</td>
<td>This parameter shall be used to define a signal value which shall be compared (CanIcomSignalOperation) with the masked CanIcomSignalMask value of the received signal (CanIcomSignalRef).</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>EcucIntegerParamDef</td>
</tr>
<tr>
<td>Range</td>
<td>0 .. 18446744073709551615</td>
</tr>
<tr>
<td>Default value</td>
<td>--</td>
</tr>
<tr>
<td>Post-Build Variant Value</td>
<td>false</td>
</tr>
<tr>
<td>Value Configuration Class</td>
<td>Pre-compile time: X All Variants</td>
</tr>
<tr>
<td>Link time</td>
<td>--</td>
</tr>
<tr>
<td>Post-build time</td>
<td>--</td>
</tr>
<tr>
<td>Scope / Dependency</td>
<td>scope: local</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SWS Item</th>
<th>ECUC_Can_00456 :</th>
</tr>
</thead>
</table>
### Name
CanIcomSignalRef

### Parent Container
CanIcomRxMessageSignalConfig

### Description
This parameter defines a reference to the signal which shall be checked additional to the message id (CanIcomMessageId). This reference is used for documentation to define which ComSignal originates this filter setting. All signals being referred by this reference shall point to the same PDU.

### Multiplicity
0..1

### Type
Reference to [ ComSignal ]

### Post-Build Variant Multiplicity
false

### Post-Build Variant Value
false

### Multiplicity Configuration Class

<table>
<thead>
<tr>
<th>Scope / Dependency</th>
<th>All Variants</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope: ECU</td>
<td>X</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

### Value Configuration Class

<table>
<thead>
<tr>
<th>Scope / Dependency</th>
<th>All Variants</th>
<th>Pre-compile time</th>
<th>Link time</th>
<th>Post-build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependency: The signal referenced by CanIcomSignalRef shall be included in a ComIPdu which matches with the current CAN Controller and the CAN Identifier (CanIcomMessageId) configured for this CanIcomRxMessage.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### No Included Containers

### 10.2.15 CanIcomWakeupCauses

#### SWS Item
ECUC_Can_00443 :

#### Container Name
CanIcomWakeupCauses

#### Description
This container contains the configuration parameters of the wakeup causes to leave the power saving mode.

#### Configuration Parameters

### Included Containers

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Multiplicity</th>
<th>Scope / Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanIcomRxMessage</td>
<td>1..*</td>
<td>This container contains the configuration parameters for the wakeup causes for matching received messages. It has to be configured as often as received messages are defined as wakeup cause. constraint: For all CanIcomRxMessage instances the Message IDs which are defined in CanIcomMessageId and in CanIcomRxMessageIdMask shall not overlap.</td>
</tr>
</tbody>
</table>
11 Not applicable requirements

[SWS_Can_00999] These requirements are not applicable to this specification.

(SRS_BSW_00170, SRS_BSW_00383, SRS_BSW_00395, SRS_BSW_00397,
SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00168,
SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426,
SRS_BSW_00427, SRS_BSW_00429, SRS_BSW_00433, SRS_BSW_00336,
SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00409, SRS_BSW_00162,
SRS_BSW_00415, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00453,
SRS_BSW_00413, SRS_BSW_00307, SRS_BSW_00447, SRS_BSW_00353,
SRS_BSW_00361, SRS_BSW_00439, SRS_BSW_00449, SRS_BSW_00378,
SRS_BSW_00359, SRS_BSW_00440, SRS_SPAL_12163, SRS_SPAL_12462,
SRS_SPAL_12068, SRS_SPAL_12064, SRS_Can_01125, SRS_Can_01126)