Document Title	Specification of CAN Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	11

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R23-11

Document Change History

Date	**Release**	**Changed by**	**Description**
2023-11-23	R23-11	AUTOSAR Release Management	• Support for selective WakeUp via CAN-Controller • Changed document name to include “CP” • Added information about automatic handle IDs to configuration • Converted to LaTeX
2022-11-24	R22-11	AUTOSAR Release Management	• CanXL requirements were added • Minor corrections / clarifications / editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Timestamp requirements were added • Removed [SWS_Can_00485] and ECUC_Can_00466 • Changed the scope of CanIndex from local to ECU global • Minor corrections / clarifications / editorial changes

▽

△

2020-11-30	R20-11	AUTOSAR Release Management	• Removed Pretended Networking • CanDrv_CONSTR_00512 was added • Updated ECUC_Can_00471 descripton • Add new parameter: CanObjectPayloadLength • A note was added to [SWS_Can_00403] • [SWS_Can_00222] was changed • Minor corrections / clarifications / editorial changes • Added Reporting of CAN Error Types chapter. Requirement [SWS_Can_91021] was added. • CanEnableSecurityEventReporting container was added
2019-11-28	R19-11	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• MCALMulticoreDistribution (CONC_639) as DRAFT • BusMirroring (CONC_634) • Header file cleanup • Replaced ChannelId with ShortName for multiple main functions ([SWS_Can_00441] and [SWS_Can_00442]) • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation

▽

△

2017-12-08	4.3.1	AUTOSAR Release Management	• Added Support to Tx/RxProcessing per Controller • Incompatible return types are corrected to E_NOT_OK and E_OK • Can_StateTransitionType is removed • Runtime error is added and Rephrased from "default error" to "development error" • [SWS_Can_00504] and [SWS_Can_00416] is modified
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
2014-10-31	4.2.1	AUTOSAR Release Management	• Full CAN FD Support (incl. Trigger Transmit) • Removed CanIf_CancelTxConfirmation • Time-out and wake up event handling • Small improvements and minor bug-fixes
2014-03-31	4.1.3	AUTOSAR Release Management	• Added new reqirements [SWS_Can_00497], [SWS_Can_00498], [SWS_Can_00499], and [SWS_Can_00496] • Modified reqirements ECUC_Can_00445, [SWS_Can_00487], [SWS_Can_00469], [SWS_Can_00475], and [SWS_Can_00479] • Removed reqirements [SWS_Can_00476], and [SWS_Can_00414]
2013-10-31	4.1.2	AUTOSAR Release Management	• Removed the 'Timing' row from the API table(s) of chapter 'Scheduled Functions' • Modified range of Can_IdType and CAN_CHANGE_BAUDRATE_SUPPORT to CAN_CHANGE_BAUDRATE_API • Editorial changes • Removed chapter(s) on change documentation

▽

△

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_SetWakeupEvent 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
2011-12-22	4.0.3	AUTOSAR Administration	• Added [SWS_Can_00461] to capture - Detection of Power ON of controller due to CAN communication • Changed Can_InitController to Can_ChangeBaudrate • Added Can_CheckBaudrate • Added sub container CanMainFunctionRWPeriods to CanGeneral • Changed CanHardwareObject container • Updated description of ECUC_Can_00321 • Changed Can_SetControllerMode in [SWS_Can_00370] to Can_Mainfunction_Mode • Added CanControllerDefaultBaudrate parameter • Updated description of [SWS_Can_00279] • Updated description of CAN321 ▽

▽

△

			△ • Added [SWS_Can_00445], [SWS_Can_00446] and [SWS_Can_00447] to capture Possible loss of CAN Wakeup • Changed "Module Short Name" (MODULENAME) to "Module Abbreviation" (MAB)
2009-12-18	4.0.1	AUTOSAR Administration	• Modified [SWS_Can_00111] to correct the "Version Checking" information • Added new requirements [SWS_Can_00435] to [SWS_Can_00440] to introduce Can_GeneralTypes.h. • Added new requirements [SWS_Can_00441] and [SWS_Can_00442] to introduce multiple poll cycles • Added new requirements [SWS_Can_00443] and [SWS_Can_00444] to provide an optional callback on every reception of a LPDU
2010-02-02	3.1.4	AUTOSAR Administration	• General improvements of requirements in preparation of CT-development. • Can_MainFunctio_Mode added to support asynchronous controller state change • Limited number of supported message objects removed • Description of CAN controller state transitions improved • Debbuging concept added • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2008-02-01	3.0.2	AUTOSAR Administration	• Table formatting corrected

▽

△

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
2007-01-24	2.1.15	AUTOSAR Administration	• File structure reworked (chapter REF _Ref158085666 \r \h) • Removed return value CAN_WAKEUP in function Can_SetControllerMode • Replaced by CAN_NOT_OK • Renamed CanIf_ControllerWakeup to CanIf_SetWakeupEvent • Reworked development errors (chapter REF _Ref182101189 \r \h) • Removed implementation specific description in Can_Write • Changed timing of cyclic functions to "fixed cyclic" • Reworked "Scope" for all configuration variables (chapter REF _Ref104709655 \r \h) • Legal disclaimer revised • Release notes added ▽

▽

△

			△ • "Advice for users" revised • "Revision Information" added
2006-05-16	2.0	AUTOSAR Administration	• Document structure adapted to common Release 2.0 SWS Template • clarified development and production error handling and function abortion • multiplexed transmission and TX cancellation • version check • configuration description according template • individual main functions for RX TX and status
2005-05-31	1.0	AUTOSAR Administration	• Initial release

Contents

1 Introduction and functional Overview 13

2 Acronyms and Abbreviations 14

2.1 Priority Inversion 15
2.2 CAN Hardware Unit 17

3 Related Documentation 18

3.1 Related specification 18

4 Constraints and assumptions 19

4.1 Limitations 19
4.2 Applicability to car domains 19

5 Dependencies to other modules 20

5.1 Static Configuration 20
5.2 Driver Services 20
5.3 System Services 20
5.4 Can module Users 21
5.5 File Structure 21

6 Requirements Tracing 22

7 Functional Specification 26

7.1 Driver Scope 26
7.2 Driver State Machine 27
7.3 CAN Controller State Machine 28
7.3.1 CAN Controller State Description 28
7.3.2 CAN Controller State Transitions 29
7.3.3 State transition caused by function Can_Init 30
7.3.4 State transition caused by function Can_SetBaudrate 30
7.3.5 State transition caused by function Can_SetControllerMode . 31
7.3.6 State transition caused by Hardware Events 33
7.3.7 State transition caused by function Can_DeInit 34
7.4 Can module/Controller Initialization 34
7.5 L-PDU transmission 35
7.5.1 Priority Inversion 36
7.5.2 Transmit Data Consistency 37
7.6 L-PDU reception 38
7.6.1 Receive Data Consistency 38
7.7 Wakeup Concept 40
7.8 CAN Controller with selective wakeup functionality 40
7.9 Notification concept 41
7.10 Reentrancy issues 41
7.11 Hardware Timestamping 42
7.12 Error classification 42

7.12.1 Development Errors 43
7.12.2 Runtime Errors 43
7.12.3 Transient Faults 43
7.12.4 Production Errors 43
7.12.5 Extended Production Errors 44
7.12.6 Return Value 44
7.13 CAN FD Support 44
7.14 CAN XL Extension 45
7.15 Reporting of CAN Error Types 45

8 API Specification 46

8.1 Imported Types 46
8.2 Type definitions 46
8.2.1 Can_ConfigType 46
8.2.2 Can_PduType 47
8.2.3 Can_IdType 47
8.2.4 Can_HwHandleType 48
8.2.5 Can_HwType 48
8.2.6 Extension to Std_ReturnType 48
8.2.7 Can_ErrorStateType 49
8.2.8 Can_ControllerStateType 49
8.2.9 Can_ErrorType 50
8.2.10 Can_TimeStampType 50
8.3 Function Definitions 51
8.3.1 Services affecting the complete hardware unit 51
8.3.1.1 Can_Init 51
8.3.1.2 Can_GetVersionInfo 52
8.3.1.3 Can_DeInit 52
8.3.2 Services affecting one single CAN Controller 53
8.3.2.1 Can_SetBaudrate 53
8.3.2.2 Can_SetControllerMode 54
8.3.2.3 Can_DisableControllerInterrupts 55
8.3.2.4 Can_EnableControllerInterrupts 57
8.3.2.5 Can_CheckWakeup 57
8.3.2.6 Can_GetControllerErrorState 58
8.3.2.7 Can_GetControllerMode 59
8.3.2.8 canGetControllerRxErrorCounter 60
8.3.2.9 Can_GetControllerTxErrorCounter 61
8.3.2.10 Can_GetCurrentTime 62
8.3.2.11 Can_EnableEgressTimeStamp 62
8.3.2.12 Can_GetEgressTimeStamp 63
8.3.2.13 Can_GetIngressTimeStamp 65
8.3.3 Services affecting a Hardware Handle 66
8.3.3.1 Can_Write 66
8.4 Call-back notifications 68
8.4.1 Call-out function 68

8.4.2 Enabling/Disabling wakeup notification 69
8.5 Scheduled functions 69
8.5.1 70
8.5.1.1 Can_MainFunction_Write 70
8.5.1.2 Can_MainFunction_Read 70
8.5.1.3 Can_MainFunction_BusOff 71
8.5.1.4 Can_MainFunction_Wakeup 72
8.5.1.5 Can_MainFunction_Mode 72
8.6 Expected Interfaces 72
8.6.1 Mandatory Interfaces 73
8.6.2 Optional Interfaces 73
8.6.3 Configurable Interfaces 74

9 Sequence diagrams 75

9.1 Interaction between Can and CanIf module 75
9.2 Wakeup sequence 75

10 Configuration specification 76

10.1 How to read this chapter 76
10.2 Containers and configuration parameters 76
10.2.1 Can 84
10.2.2 CanGeneral 84
10.2.3 CanController 90
10.2.4 CanControllerBaudrateConfig 95
10.2.5 CanControllerFdBaudrateConfig 97
10.2.6 CanPartialNetwork 100
10.2.7 CanPnFrameDataMaskSpec 101
10.2.8 CanHardwareObject 102
10.2.9 CanHwFilter 107
10.2.10 CanConfigSet 108
10.2.11 CanMainFunctionRWPeriods 109
10.2.12 CanTTController 109
10.2.13 CanTTHardwareObjectTrigger 116
10.2.14 CanXLGeneral 119
10.2.15 CanXLController 119
10.2.16 CanXLHardwareObject 121
10.2.17 CanXLHwFilter 123
10.2.18 CanXLBaudrateConfig 124
10.2.19 CanXLEthEgressFifo 127
10.2.20 CanXLEthIngressFifo 129

11 Not applicable requirements 131

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 [1].

2 Acronyms and Abbreviations

Abbreviation / Acronym:	Description:
CAN controller	A CAN controller serves exactly one physical channel.
CAN Hardware Unit	A CAN Hardware Unit may consists 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.
CAN L-PDU	Data Link Layer Protocol Data Unit. Consists of Identifier, Data Length and Data (SDU). (see[2])
CAN L-SDU	Data Link Layer Service Data Unit. Data that is transported inside the L-PDU. (see[2])
DLC	Data Length Code (part of CAN message describes the SDU length)
Hardware Object	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.
Hardware Receive Handle (HRH)	The Hardware Receive Handle (HRH) is defined and provided by the CAN Driver. Each HRH typically represents just one hardware object. The HRH can be used to optimize software filtering.
Hardware Transmit Handle (HTH)	The Hardware Transmit Handle (HTH) is defined and provided by the CAN Driver. Each HTH typically represents just one or multiple hardware objects that are configured as hardware transmit buffer pool.
Inner Priority Inversion	Transmission of a high-priority L-PDU is prevented by the presence of a pending low-priority L-PDU in the same transmit hardware object.
ISR	Interrupt Service Routine
L-PDU Handle	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.
MCAL	Microcontroller Abstraction Layer
Outer Priority Inversion	A time gap occurs between two consecutive transmit L-PDUs. In this case a lower priority L-PDU from another node can prevent sending the own higher priority L-PDU. Here the higher priority L-PDU cannot participate in arbitration during network access because the lower priority L-PDU already won the arbitration.
Physical Channel	A physical channel represents an interface from a CAN controller to the CAN Network. Different physical channels of the CAN hardware unit may access different networks.
Priority	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.
SFR	Special Function Register. Hardware register that controls the controller behavior.
SPAL	Standard Peripheral Abstraction Layer

Table 2.1: Acronyms and Abbreviations

2.1 Priority Inversion

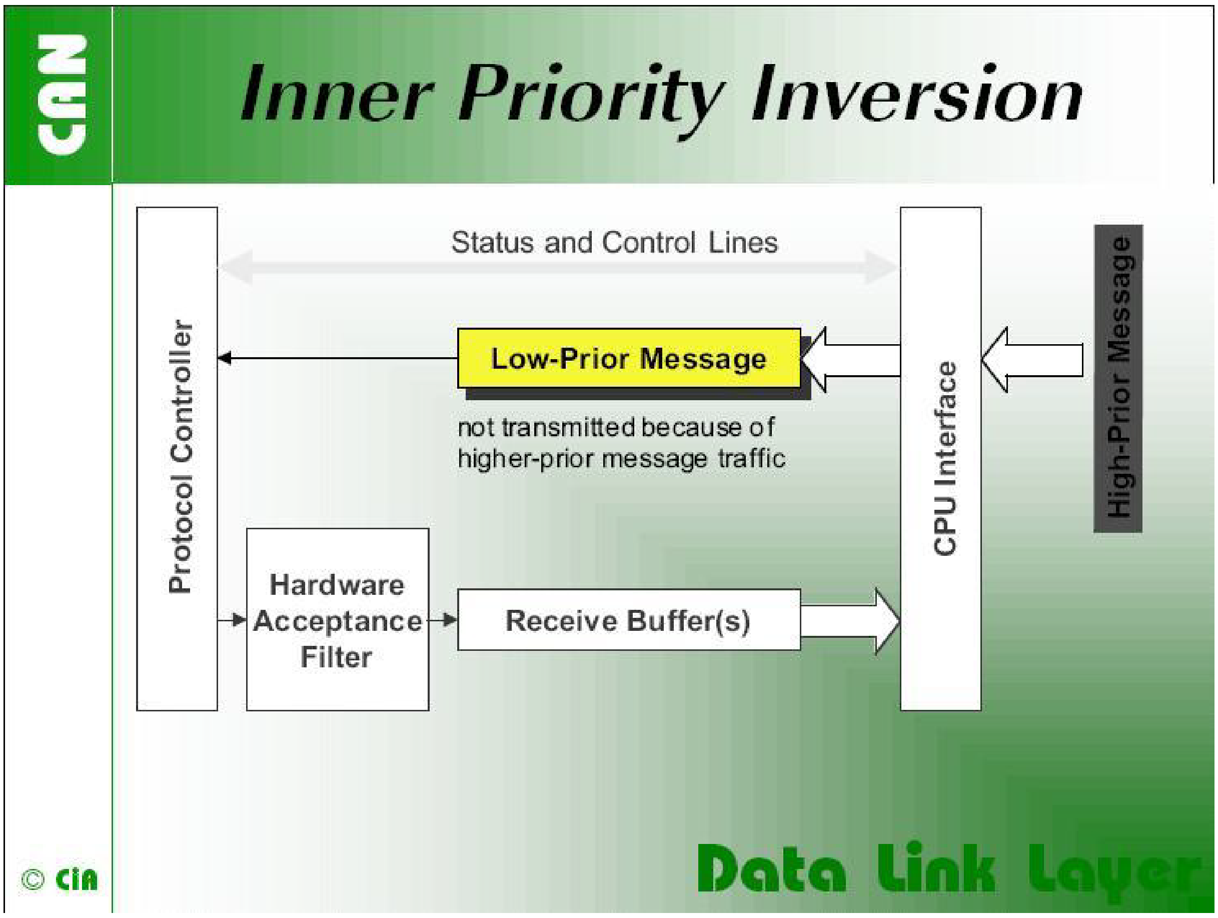

Figure 2.1

"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]

[1] Picture and text by CiA (CAN in Automation)

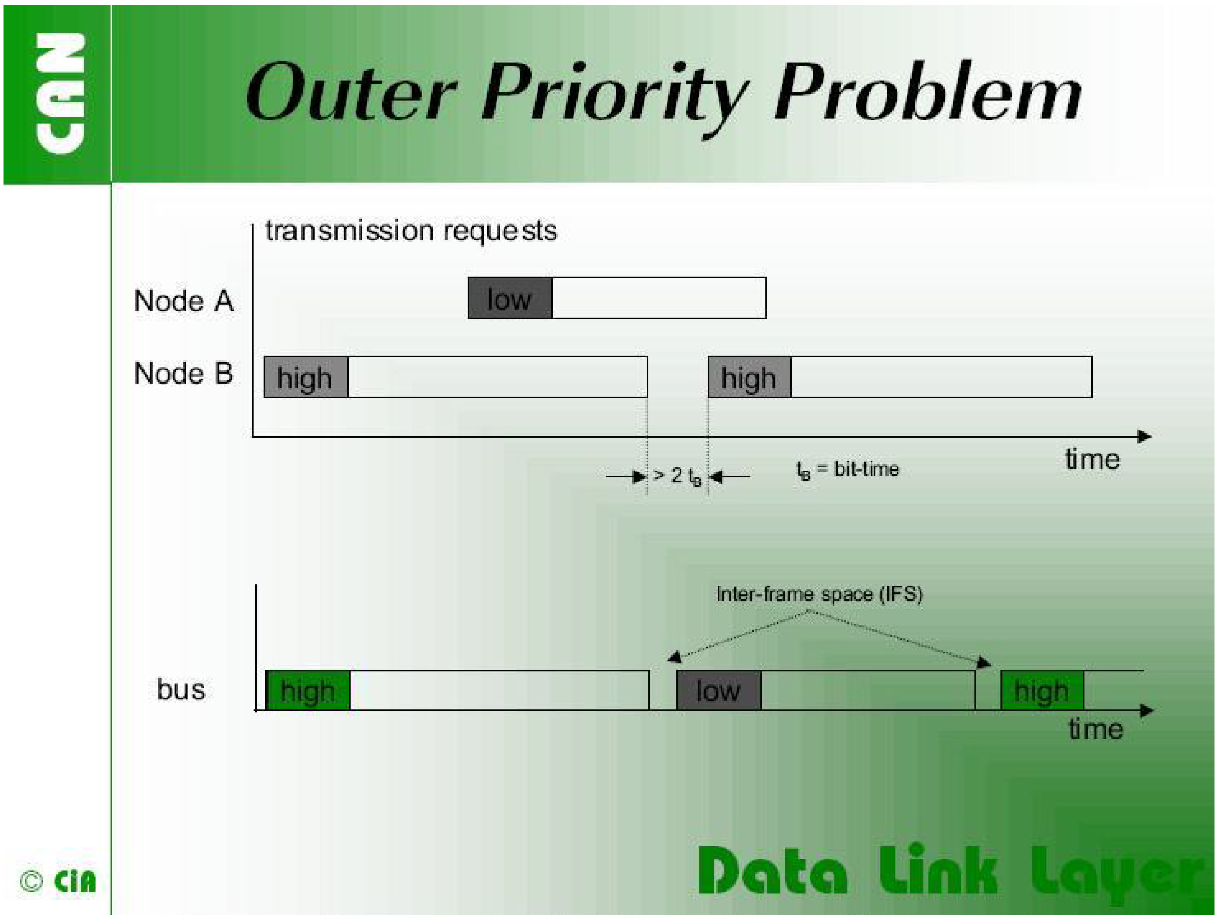

Figure 2.2

"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:

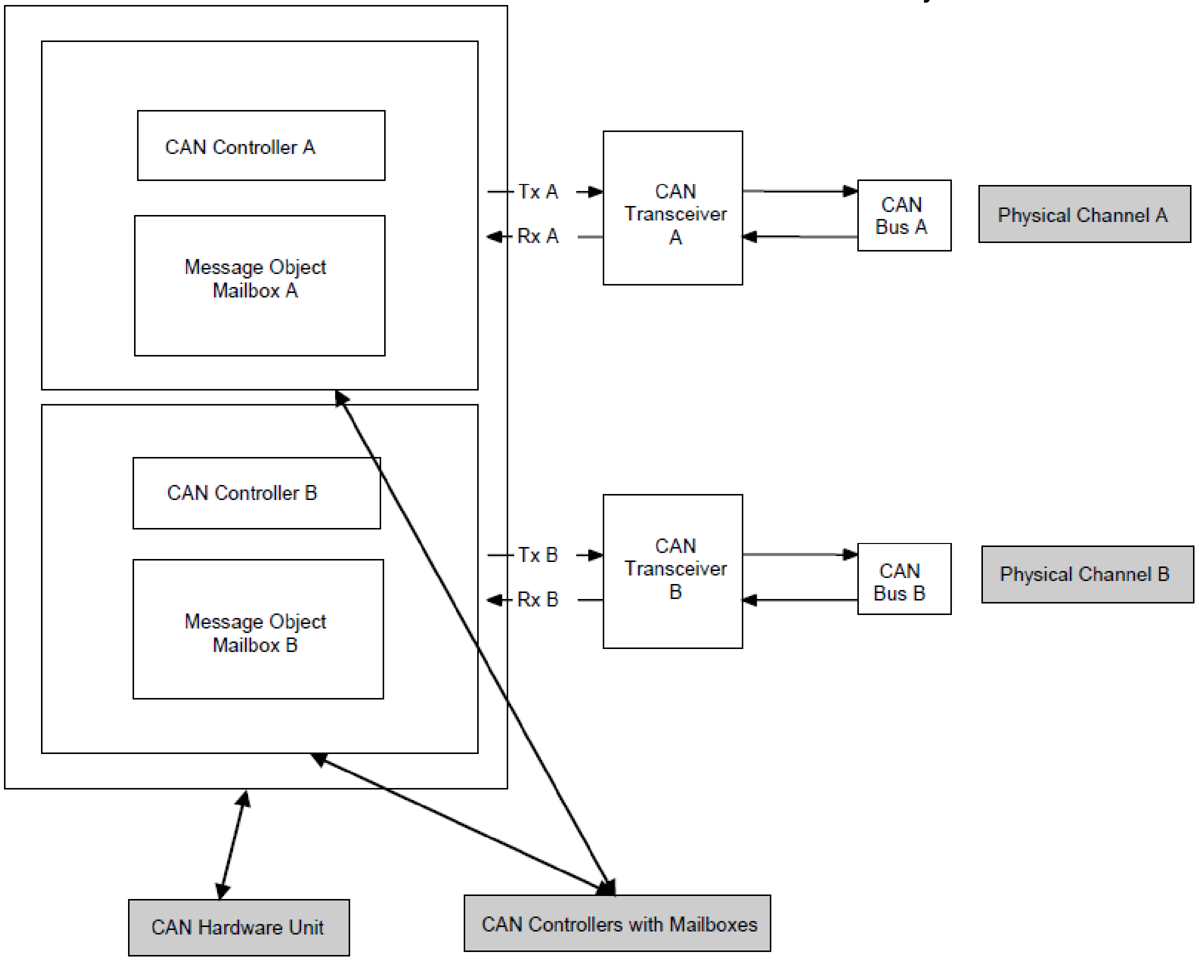

Figure 2.3

3 Related Documentation

[1] Specification of CAN Interface
AUTOSAR_CP_SWS_CANInterface

[2] IEC:The Basic Model, IEC Norm

[3] List of Basic Software Modules
AUTOSAR_CP_TR_BSWModuleList

[4] Specification of MCU Driver
AUTOSAR_CP_SWS_MCUDriver

[5] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[6] General Requirements on SPAL
AUTOSAR_CP_SRS_SPALGeneral

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

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

[9] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

3.1 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3] (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 [4]) 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[3], 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 [5].

[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. 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

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

6 Requirements Tracing

Requirement	Description	Satisfied by
[RS_Ids_00810]	Basic SW security events	[SWS_Can_91022] [SWS_Can_91023] [SWS_Can_91024]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_Can_00238] [SWS_Can_00242]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Can_00079]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Can_00250]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_Can_00022]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_Can_00033]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_Can_00024]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Can_00079]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_Can_00079]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_Can_00079]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Can_00214] [SWS_Can_00231] [SWS_Can_00232] [SWS_Can_00233]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Can_00026] [SWS_Can_00513] [SWS_Can_00514] [SWS_Can_00518] [SWS_Can_00519] [SWS_Can_91006] [SWS_Can_91007] [SWS_Can_91017] [SWS_Can_91018]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_Can_00079]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_Can_00039] [SWS_Can_00104]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_Can_91002]
[SRS_BSW_00337]	Classification of development errors	[SWS_Can_00026] [SWS_Can_00104]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Can_00021]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_Can_00077]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_Can_00506]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Can_00223]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Can_00089] [SWS_Can_00506] [SWS_Can_91011] [SWS_Can_91012]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_Can_00031]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_Can_00271] [SWS_Can_00364]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[SWS_Can_00239]
[SRS_BSW_00385]	List possible error notifications	[SWS_Can_00104]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Can_00089]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Can_00021]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Can_00021]
[SRS_BSW_00406]	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	[SWS_Can_00103] [SWS_Can_00512] [SWS_Can_00517] [SWS_Can_91005] [SWS_Can_91016]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Can_00223]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_Can_91005] [SWS_Can_91016]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_Can_00110]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_Can_00031] [SWS_Can_00108] [SWS_Can_00109] [SWS_Can_00112]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_Can_00291]
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	[SWS_Can_00506]
[SRS_Can_01005]	The CAN Interface shall perform a check for correct DLC of received PDUs	[SWS_Can_00218]
[SRS_Can_01041]	The CAN Driver shall implement an interface for initialization	[SWS_Can_00245] [SWS_Can_00246]
[SRS_Can_01042]	The CAN Driver shall support dynamic selection of configuration sets	[SWS_Can_00062]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01043]	The CAN Driver shall provide a service to enable/disable interrupts of the CAN Controller.	[SWS_Can_00049] [SWS_Can_00050]
[SRS_Can_01045]	The CAN Driver shall offer a reception indication service.	[SWS_Can_00279] [SWS_Can_00396]
[SRS_Can_01049]	The CAN Driver shall provide a dynamic transmission request service	[SWS_Can_00212] [SWS_Can_00213] [SWS_Can_00214]
[SRS_Can_01051]	The CAN Driver shall provide a transmission confirmation service	[SWS_Can_00016]
[SRS_Can_01053]	The CAN Driver shall provide a service to change the CAN controller mode.	[SWS_Can_00017] [SWS_Can_91010]
[SRS_Can_01054]	The CAN Driver shall provide a notification for controller wake-up events	[SWS_Can_00235] [SWS_Can_00271] [SWS_Can_00364]
[SRS_Can_01055]	CAN Driver shall provide a notification for bus-off state	[SWS_Can_00020] [SWS_Can_00234]
[SRS_Can_01059]	The CAN Driver shall guarantee data consistency of received L-PDUs	[SWS_Can_00011] [SWS_Can_00012]
[SRS_Can_01060]	The CAN driver shall not recover from bus-off automatically	[SWS_Can_00272] [SWS_Can_00273] [SWS_Can_00274]
[SRS_Can_01062]	Each event for each CAN Controller shall be configurable to be detected by polling or by an interrupt	[SWS_Can_00007]
[SRS_Can_01122]	The CAN driver shall support the situation where a wakeup by bus occurs during the same time the transition to standby/sleep is in progress	[SWS_Can_00048]
[SRS_Can_01130]	Receive Status Interface of CAN Interface	[SWS_Can_00506]
[SRS_Can_01132]	The CAN driver shall be able to detect notification events message object specific by CAN-Interrupt and polling	[SWS_Can_00099]
[SRS_Can_01134]	The CAN Driver shall support multiplexed transmission	[SWS_Can_00277] [SWS_Can_00401] [SWS_Can_00402] [SWS_Can_00403]
[SRS_Can_01135]	It shall be possible to configure one or several TX Hardware Objects	[SWS_Can_00100]
[SRS_Can_01139]	The CAN Interface and Driver shall offer a CAN Controller specific interface for initialization	[SWS_Can_00062]
[SRS_Can_01147]	The CAN Driver shall not support remote frames	[SWS_Can_00236] [SWS_Can_00237]
[SRS_Can_01160]	Padding of bytes due to discrete CAN FD DLC	[SWS_Can_00502]
[SRS_Can_01162]	CAN Interface shall support classic CAN and CAN FD frames	[SWS_Can_00501]
[SRS_Can_01166]	The CAN Driver shall implement an interface for de-initialization	[SWS_Can_91002] [SWS_Can_91009] [SWS_Can_91010]
[SRS_Can_01167]	The CAN Driver shall provide a function to return the current CAN controller error state	[SWS_Can_91008]
[SRS_Can_01170]	The CAN Driver shall provide a function to return the current CAN controller Rx and Tx error counters	[SWS_Can_00515] [SWS_Can_00520]

▽

△

Requirement	Description	Satisfied by
[SRS_Can_01181]	If partial networking is used, the ECU shall secure that the first message on the bus is the wakeup frame.	[SWS_CAN_91025] [SWS_CAN_91026] [SWS_CAN_91027] [SWS_CAN_91028] [SWS_CAN_91029]
[SRS_SPAL_00157]	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	[SWS_Can_00026] [SWS_Can_00031] [SWS_Can_00108] [SWS_Can_00109] [SWS_Can_00112]
[SRS_SPAL_12056]	All driver modules shall allow the static configuration of notification mechanism	[SWS_Can_00235]
[SRS_SPAL_12057]	All driver modules shall implement an interface for initialization	[SWS_Can_00245] [SWS_Can_00246]
[SRS_SPAL_12063]	All driver modules shall only support raw value mode	[SWS_Can_00059] [SWS_Can_00060]
[SRS_SPAL_12067]	All driver modules shall set their wake-up conditions depending on the selected operation mode	[SWS_Can_00257]
[SRS_SPAL_12069]	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	[SWS_Can_00271] [SWS_Can_00364]
[SRS_SPAL_12075]	All drivers with random streaming capabilities shall use application buffers	[SWS_Can_00011]
[SRS_SPAL_12077]	All drivers shall provide a non blocking implementation	[SWS_Can_00372]
[SRS_SPAL_12092]	The driver's API shall be accessed by its handler or manager	[SWS_Can_00058]
[SRS_SPAL_12125]	All driver modules shall only initialize the configured resources	[SWS_Can_00053]
[SRS_SPAL_12129]	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	[SWS_Can_00033]
[SRS_SPAL_12169]	All driver modules that provide different operation modes shall provide a service for mode selection	[SWS_Can_00017]
[SRS_SPAL_12263]	The implementation of all driver modules shall allow the configuration of specific module parameter types at link time	[SWS_Can_00021]
[SRS_SPAL_12265]	Configuration data shall be kept constant	[SWS_Can_00021]
[SRS_SPAL_12448]	All driver modules shall have a specific behavior after a development error detection	[SWS_Can_00089] [SWS_Can_00091]
[SRS_SPAL_12461]	Specific rules regarding initialization of controller registers shall apply to all driver implementations	[SWS_Can_00407]
[SRS_SPAL_12463]	The register initialization settings shall be combined and forwarded	[SWS_Can_00024]

Table 6.1: RequirementsTracing

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 [6]).

[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 [6].⌋*(SRS_BSW_00007, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, 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 [1] 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. 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)*

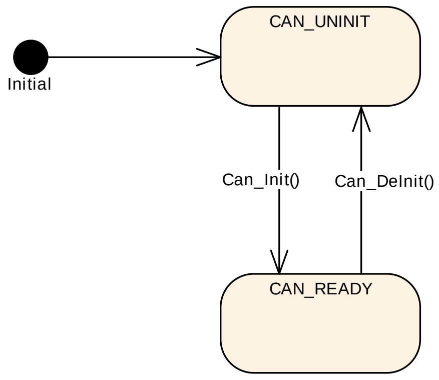

Figure 7.1

[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_SetController-Mode` 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.
The behavior for invalid transitions in production code is undefined. 7.2 shows all valid state transitions.

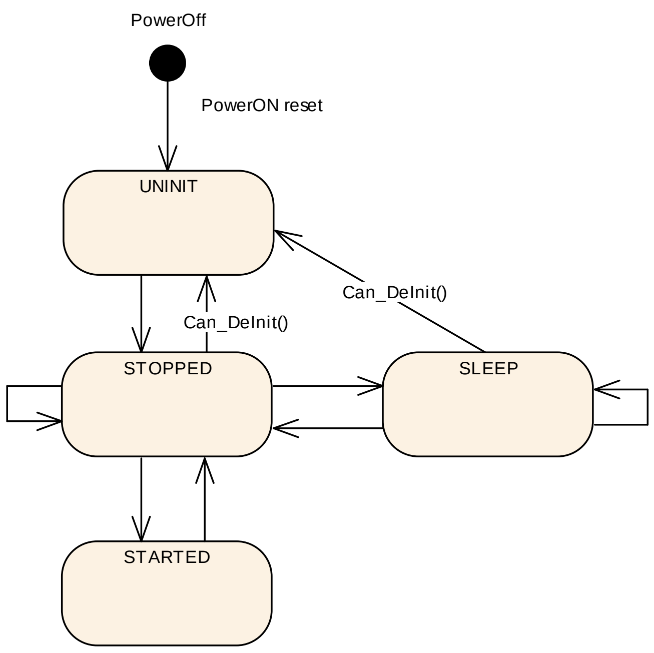

Figure 7.2

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_Wakeup 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_Wakeup shall continue to poll the flag.⌋*(SRS_SPAL_12077)*

[SWS_Can_00373] ⌈The function Can_MainFunction_Wakeup 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_00282] ⌈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.⌋*()*

[SWS_Can_00266] ⌈If the CAN HW does support a sleep mode, the function `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 [SWS_Can_00398].⌋*()*

[SWS_Can_00290] ⌈If the CAN HW does not support a sleep mode, the function `Can_SetControllerMode`(CAN_CS_SLEEP) shall set the CAN controller to the logical sleep mode.⌋*()*

[SWS_Can_00405] ⌈This logical sleep mode shall left only, if function Can_SetControllerMode(CAN_CS_STOPPED) is called.⌋*()*

[SWS_Can_00411] ⌈When the function 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 [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 -> STOPPED
- triggered by incoming L-PDUs
- The ECU Statemanager module is notified with the function EcuM_CheckWakeup

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

[SWS_Can_00270] ⌈On hardware wakeup (triggered by a wake-up event from CAN bus), the CAN controller shall transition into the state STOPPED.⌋*()*

[SWS_Can_00271] ⌈On hardware wakeup (triggered by a wake-up event from CAN bus), the Can module shall call the function EcuM_CheckWakeup either in interrupt context or in the context of Can_MainFunction_Wakeup.⌋*(SRS_BSW_00375, SRS_SPAL_12069, SRS_Can_01054)*

[SWS_Can_00269] ⌈The Can module shall not further process the L-PDU that caused a wake-up.⌋*()*

[SWS_Can_00048] ⌈In case of a CAN bus wake-up during sleep transition, the function Can_SetControllerMode(CAN_CS_STOPPED) shall return E_NOT_OK.⌋*(SRS_Can_01122)*

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

[SWS_Can_00020] ⌈

- STARTED -> STOPPED
- triggered by hardware if the CAN controller reaches bus-off state
- The CanIf module is notified with the function CanIf_ControllerBusOff after STOPPED state is reached referring to the corresponding CAN controller with the abstract CanIf ControllerId.

⌋*(SRS_Can_01055)*

[SWS_Can_00272] ⌈After bus-off detection, the CAN controller shall transition to the state STOPPED and the Can module shall ensure that the CAN controller doesn't participate on the network anymore.⌋*(SRS_Can_01060)*

[SWS_Can_00273] ⌈After bus-off detection, the Can module shall cancel still pending messages.⌋*(SRS_Can_01060)*

[SWS_Can_00274] ⌈The Can module shall disable or suppress automatic bus-off recovery.⌋*(SRS_Can_01060)*

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)*

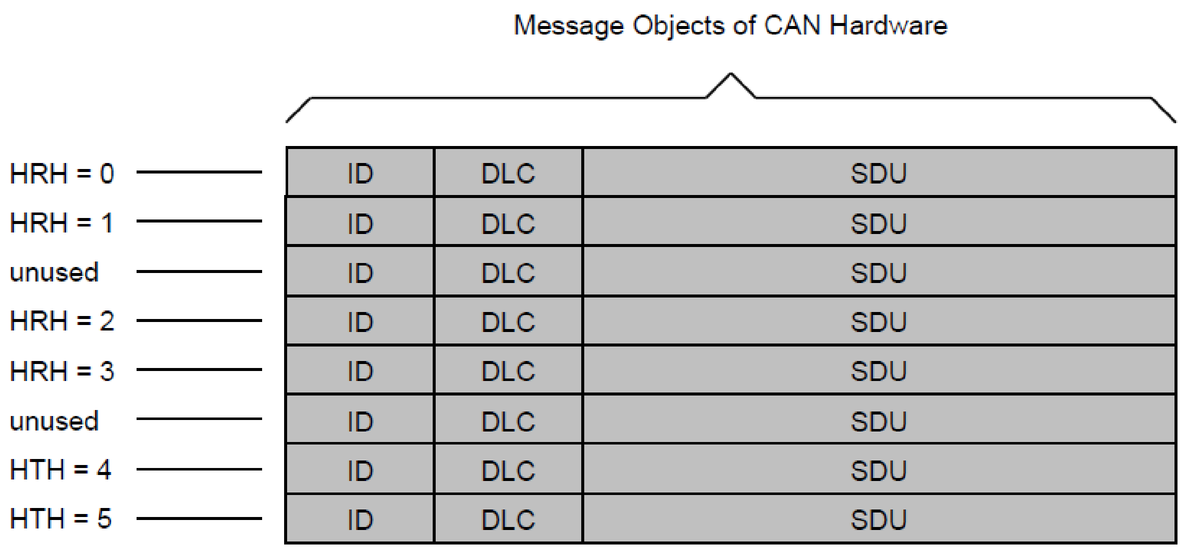

Figure 7.3: Example of assignment of HTHs and HRHs to the Hardware Objects. The numbering of HTHs and HRHs are implementation specific. The chosen numbering is only an example.

[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 can be used to avoid outer/inner 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.

⌋*(SRS_Can_01134)*

[SWS_Can_00403] ⌈The Can module shall support multiplexed transmission for devices, which send L-PDUs in order of L-PDU priority.⌋*(SRS_Can_01134)*

Note: Ordering of L-PDUs by priority avoids inner priority inversion of the L-PDUs assigned to a Basic-CAN configured for multiplexed transmission. Another possibility to avoid inner priority inversion is the configuration of all HTHs to be Full-CAN if the CAN hardware is able to prioritize upon transmission using the CAN ID or related priority field.

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

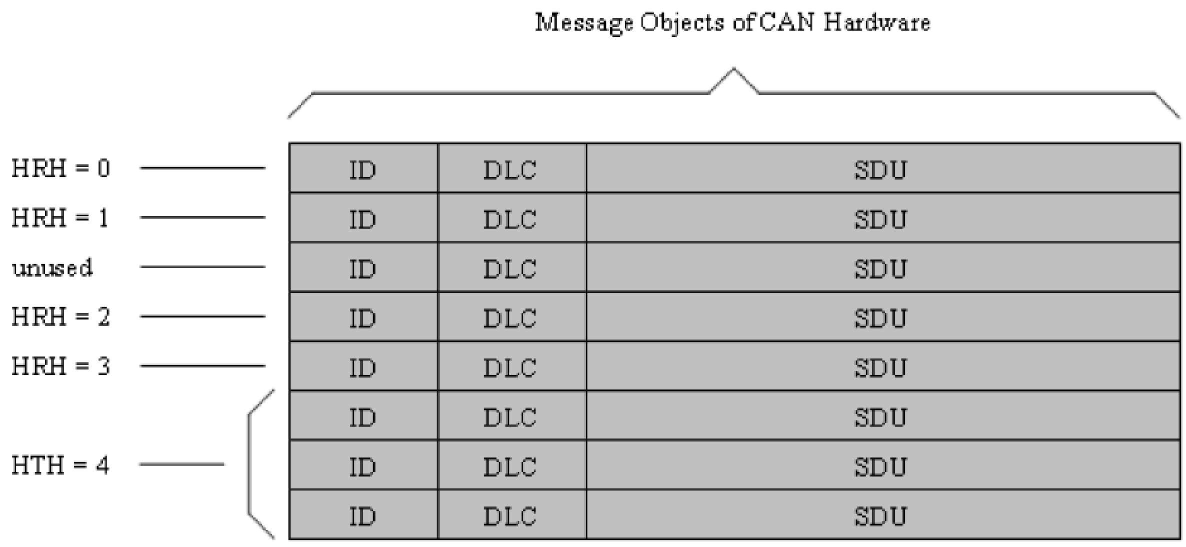

Figure 7.4: Example of assignment of HTHs and HRHs to the Hardware Objects with multiplexed transmission. The numbering of HTHs and HRHs are implementation specific. The chosen numbering is only an example.

[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`).⌋*(SRS_SPAL_12075, SRS_Can_01059)*

7.5.2 Transmit Data Consistency

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".⌋*()*

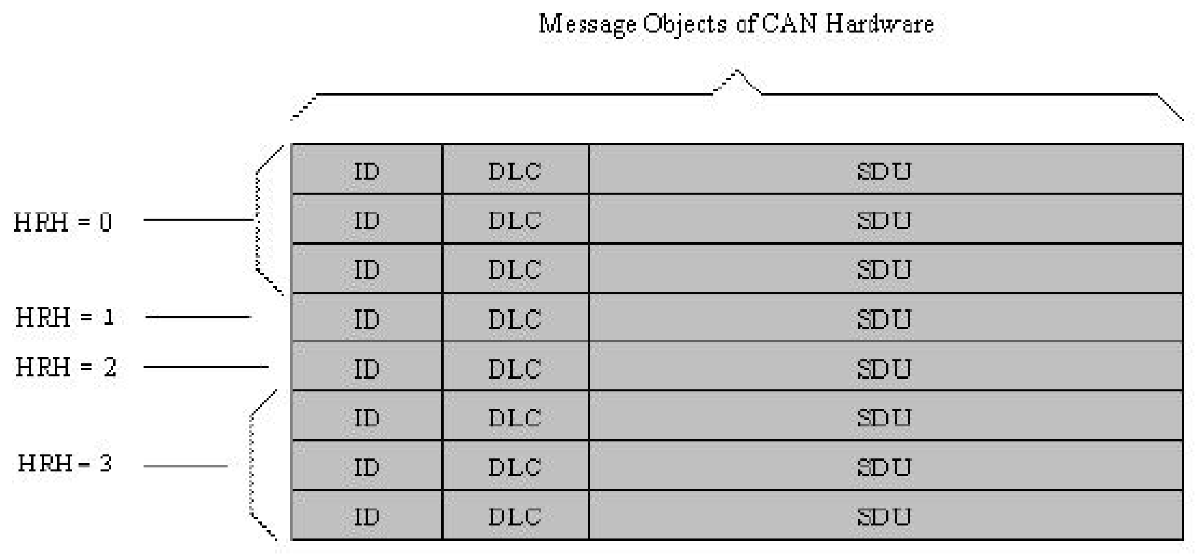

Figure 7.5: Example of assignment of same HRHs to multiple Hardware Objects The chosen numbering is only an example.

[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 fasted 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 [5].

7.8 CAN Controller with selective wakeup functionality

This section describes requirements for CAN controller with selective wakeup functionality.
Partial Networking is a state in a CAN system where some nodes are in low power mode while other nodes are communicating. This reduces the power consumption by the entire network. Nodes in the low-power modes are woken up by predefined wakeup frames.
CAN Controller which support selective wakeup can be woken up by predefined wakeup frames.

[SWS_Can_00601] ⌈If selective wakeup is supported by the CAN controller hardware, it shall be indicated with the configuration parameter CanHwPnSupport.⌋*()*

[SWS_Can_00602] ⌈If selective wakeup is supported, CAN controller shall be configured to wake up on a particular CAN frame or a group of CAN frames using the parameters CanPnFrameCanId, CanPnFrameCanIdMask and CanPnFrameDataMask.⌋
()

7.9 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`.

7.10 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 reentrant 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.

7.11 Hardware Timestamping

Hardware-based timestamping, if supported by the CAN controller, can be used e.g. to enhance the precision of a synchronized time-base on CAN. The following CAN driver APIs are provided, if hardware-based timestamping is supported:

- `Can_GetCurrentTime`
- `Can_EnableEgressTimeStamp`
- `Can_GetEgressTimeStamp`
- `Can_GetIngressTimeStamp`

Those APIs need to be enabled by the configuration parameter `CanGlobalTimeSupport`.

The hardware-based timestamping function of a CAN controller shall provide a free-running counter that is used to take the timestamps of CAN message reception and transmission. A free-running counter is a counter that counts up and overflows to zero after reaching its specified maximum value. It is specified in the CiA 603 standard that the free-running counter counts clock cycles; the resolution shall be at least 1 µs and at most 1 ns. It is highly recommended to provide 32-bit time-stamp registers and a 32-bit counter.

The timestamp for transmitted and received CAN messages is captured when the CAN frame is considered valid. Details are given in the CiA 603 standard.

7.12 Error classification

Section 7.11 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

[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)*

7.12.1 Development Errors

[SWS_Can_91019] Definiton of development errors in module Can ⌈

Type of error	*Related error code*	*Error value*
API Service called with wrong parameter	CAN_E_PARAM_POINTER	0x01
API Service called with wrong parameter	CAN_E_PARAM_HANDLE	0x02
API Service called with wrong parameter	CAN_E_PARAM_DATA_LENGTH	0x03
API Service called with wrong parameter	CAN_E_PARAM_CONTROLLER	0x04
API Service used without initialization	CAN_E_UNINIT	0x05
Invalid transition for the current mode	CAN_E_TRANSITION	0x06
Parameter Baudrate has an invalid value	CAN_E_PARAM_BAUDRATE	0x07
Invalid configuration set selection	CAN_E_INIT_FAILED	0x09
API service called with invalid PDU ID	CAN_E_PARAM_LPDU	0x0A

⌋*()*

7.12.2 Runtime Errors

[SWS_Can_91020] Definiton of runtime errors in module Can ⌈

Type of error	*Related error code*	*Error value*
Received CAN message is lost	CAN_E_DATALOST	0x01

⌋*()*

[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)*

7.12.3 Transient Faults

There are no transient faults.

7.12.4 Production Errors

There are no productions errors.

7.12.5 Extended Production Errors

There are no extended production errors.

7.12.6 Return Value

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.

7.13 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 10.2.5) 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 10.2.5).
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_IdType8.2.3, [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 10.2.5). 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.

7.14 CAN XL Extension

CAN/CAN-FD are proven in use, affordable and well distributed communication protocols with the respective communication stacks already specified within AUTOSAR.
Within the automotive industry there is a constant trend to increase communication bandwidth to cope with the complexity of modern E/E architectures. Having a lowcost, robust bus system that also follows this trend is clearly seen as a beneficial addition to the AUTOSAR standard. Therefore, CAN XL is introduced (see [7],[8]).

The goal is that CAN XL will help bridge the gap between current CAN implementations and current 100 Mbit Ethernet solutions. On the same network segment, both CAN 2.0/FD/XL and Ethernet traffic can coexist. Baudrate is not fixed to 10 Mbit like at 10BASE-T1S but can be adjusted flexible up to 20 Mbit/s. In addition, a payload up to 2048 bytes is possible.

CAN XL has a minimal impact on existing AUTOSAR Modules but still brings benefit of new properties.
Using the newly introduced CAN XL Driver it is still possible to send CAN 2.0 and CAN FD Frames without any changes.

As CAN XL Driver is implemented as an extension for the existing CAN Driver (with new document AUTOSAR_CP_SWS_CANXLDriver.pdf), non CAN XL hardware will still use basic CAN Driver implementation.

The CAN XL Driver is an extension of CAN Driver and introduces an additional API to support CAN XL Frames and Ethernet communication (see AUTOSAR_CP_SWS_CANXLDriver.pdf for further details).

7.15 Reporting of CAN Error Types

[SWS_Can_91022] ⌈If the CanEnableSecurityEventReportingtrue and CanDrv detects a CanErrorType in the range of 0x1-0xB, then CanDrv shall call CanIf_ErrorNotification with the ControllerId and the CanError as parameters.⌋*(RS_Ids_00810)*

[SWS_Can_91024] ⌈If no of the predefined Can_ErrorType values matches to the error provided by the CAN hardware, the CAN driver shall not report the error to the CanIf.⌋ *(RS_Ids_00810)*

[SWS_Can_91023] ⌈If the CanEnableSecurityEventReportingtrue and CanDrv detects a transition to error state passive, then CanDrv shall call CanIf_ControllerErrorStatePassive with the ControllerId and the values for the Rx and Tx error counters.⌋*(RS_Ids_00810)*

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 modules are listed:

[SWS_Can_00222] Definition of imported datatypes of module Can ⌈

Module	***Header File***	***Imported Type***
ComStack_Types	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
EcuM	EcuM.h	EcuM_WakeupSourceType
Icu	Icu.h	Icu_ChannelType
Os	Os.h	StatusType
	Os.h	TickRefType
	Os.h	TickType
	Rte_Os_Type.h	CounterType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋*()*

8.2 Type definitions

8.2.1 Can_ConfigType

[SWS_Can_00413] Definition of datatype Can_ConfigType ⌈

Name	Can_ConfigType
Kind	Structure
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.
Available via	Can.h

⌋*()*

8.2.2 Can_PduType

[SWS_Can_00415] Definition of datatype Can_PduType ⌈

Name	Can_PduType	
Kind	Structure	
Elements	swPduHandle	
	Type	PduIdType
	Comment	–
	length	
	Type	uint8
	Comment	–
	id	
	Type	Can_IdType
	Comment	–
	sdu	
	Type	uint8*
	Comment	–
Description	This type unites PduId (swPduHandle), SduLength (length), SduData (sdu), and CanId (id) for any CAN L-SDU.	
Available via	Can_GeneralTypes.h	

⌋*()*

8.2.3 Can_IdType

[SWS_Can_00416] Definition of ImplementationDataType Can_IdType ⌈

Name	Can_IdType		
Kind	Type		
Derived from	uint32		
Range	Standard32Bit	0..0x400007FF	0..0x400007FF
	Extended32Bit	0..0xDFFFFFFF	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 11 CAN FD frame with Extended CAN ID		
Variation	–		
Available via	Can_GeneralTypes.h		

⌋*()*

8.2.4 Can_HwHandleType

[SWS_Can_00429] Definition of datatype Can_HwHandleType ⌈

Name	Can_HwHandleType		
Kind	Type		
Derived from	***Basetype***	***Variation***	
	uint16	–	
	uint8	–	
Range	Standard	0..0x0FF	0..0x0FF
	Extended	0..0xFFFF	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.		
Available via	Can_GeneralTypes.h		

⌋()

8.2.5 Can_HwType

[SWS_CAN_00496] Definition of datatype Can_HwType ⌈

Name	Can_HwType	
Kind	Structure	
Elements	CanId	
	Type	Can_IdType
	Comment	Standard/Extended CAN ID of CAN L-PDU
	Hoh	
	Type	Can_HwHandleType
	Comment	ID of the corresponding Hardware Object Range
	ControllerId	
	Type	uint8
	Comment	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.	
Available via	Can_GeneralTypes.h	

⌋()

8.2.6 Extension to Std_ReturnType

[SWS_Can_00039] Definition of Std_ReturnType-extension for module Can ⌈

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()
Available via	Can_GeneralTypes.h

⌋*(SRS_BSW_00331)*

8.2.7 Can_ErrorStateType

[SWS_Can_91003] Definition of datatype Can_ErrorStateType ⌈

Name	Can_ErrorStateType		
Kind	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 communication.
Description	Error states of a CAN controller.		
Available via	Can_GeneralTypes.h		

⌋*()*

8.2.8 Can_ControllerStateType

[SWS_Can_91013] Definition of datatype Can_ControllerStateType ⌈

Name	Can_ControllerStateType		
Kind	Enumeration		
Range	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.
Description	States that are used by the several ControllerMode functions.		
Available via	Can_GeneralTypes.h		

⌋*()*

8.2.9 Can_ErrorType

[SWS_Can_91021] Definition of datatype Can_ErrorType ⌈

Name	Can_ErrorType		
Kind	Enumeration		
Range	CAN_ERROR_BIT_MONITORING1	0x01	A 0 was transmitted and a 1 was read back
	CAN_ERROR_BIT_MONITORING0	0x02	A 1 was transmitted and a 0 was read back
	CAN_ERROR_BIT	0x03	The HW reports a CAN bit error but cannot report distinguish between CAN_ERROR_BIT_MONITORING1 and CAN_ERROR_BIT_MONITORING0
	CAN_ERROR_CHECK_ACK_FAILED	0x04	Acknowledgement check failed
	CAN_ERROR_ACK_DELIMITER	0x05	Acknowledgement delimiter check failed
	CAN_ERROR_ARBITRATION_LOST	0x06	The sender lost in arbitration.
	CAN_ERROR_OVERLOAD	0x07	CAN overload detected via an overload frame. Indicates that the receive buffers of a receiver are full.
	CAN_ERROR_CHECK_FORM_FAILED	0x08	Violations of the fixed frame format
	CAN_ERROR_CHECK_STUFFING_FAILED	0x09	Stuffing bits not as expected
	CAN_ERROR_CHECK_CRC_FAILED	0xA	CRC failed
	CAN_ERROR_BUS_LOCK	0xB	Bus lock (Bus is stuck to dominant level)
Description	The enumeration represents a superset of CAN Error Types which typical CAN HW is able to report. That means not all CAN HW will be able to support the complete set.		
Available via	Can_GeneralTypes.h		

⌋*()*

8.2.10 Can_TimeStampType

[SWS_CAN_91029]{DRAFT} **Definition of datatype Can_TimeStampType** ⌈

Name	Can_TimeStampType (draft)	
Kind	Structure	
Elements	nanoseconds	
	Type	uint32
	Comment	Nanoseconds part of the time
	seconds	
	Type	uint32
	Comment	Seconds part of the time

▽

△

Description	Variables of this type are used to express time stamps based on relative time. Value range: * Seconds: 0 .. 4.294.967.295 s (circa 136 years) * Nanoseconds: 0 .. 999.999.999 ns **Tags:** atp.Status=draft
Available via	Can_GeneralTypes.h

⌋*(SRS_Can_01181)*

8.3 Function Definitions

8.3.1 Services affecting the complete hardware unit

8.3.1.1 Can_Init

[SWS_Can_00223] Definition of API function Can_Init ⌈

Service Name	Can_Init	

Syntax	`void Can_Init (`
	`  const Can_ConfigType* Config`
	`)`	

Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Config	Pointer to driver configuration.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This function initializes the module.	
Available via	Can.h	

⌋*(SRS_BSW_00358, SRS_BSW_00414)*

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_00408] ⌈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

[SWS_Can_00224] Definition of API function Can_GetVersionInfo ⌈

Service Name	Can_GetVersionInfo	
Syntax	`void Can_GetVersionInfo ( Std_VersionInfoType* versioninfo )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module.
Return value	None	
Description	This function returns the version information of this module.	
Available via	Can.h	

⌋*()*

[SWS_Can_00177] ⌈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

[SWS_Can_91002] Definition of API function Can_DeInit ⌈

Service Name	Can_DeInit
Syntax	`void Can_DeInit ( void )`
Service ID [hex]	0x10
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This function de-initializes the module.
Available via	Can.h

⌋*(SRS_Can_01166, SRS_BSW_00336)*

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

Caveat: Caller of the `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] Definition of API function Can_SetBaudrate ⌈

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 CanController BaudRateConfigID)
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Service request accepted, setting of (new) baud rate started `E_NOT_OK`: Service request not accepted
Description	This service shall set the baud rate configuration of the CAN controller. Depending on necessary baud rate modifications the controller might have to reset.	
Available via	Can.h	

⌋*()*

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] Definition of API function Can_SetControllerMode ⌈

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.	
Available via	Can.h	

⌋*()*

[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.⌋*()*

[SWS_Can_00196] ⌈The function Can_SetControllerMode shall enable interrupts that are needed in the new state.⌋*()*

[SWS_Can_00425] ⌈Enabling of CAN interrupts shall not be executed, when CAN interrupts have been disabled by function Can_DisableControllerInterrupts.⌋*()*

[SWS_Can_00197] ⌈The function Can_SetControllerMode shall disable interrupts that are not allowed in the new state.⌋*()*

[SWS_Can_00426] ⌈Disabling of CAN interrupts shall not be executed, when CAN interrupts have been disabled by function Can_DisableControllerInterrupts.⌋*()*

[SWS_Can_00198] ⌈If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can_SetControllerMode shall raise development error CAN_E_UNINIT and return E_NOT_OK.⌋*()*

[SWS_Can_00199] ⌈If development error detection for the Can module is enabled: if the parameter is out of range, the function Can_SetControllerMode shall raise development error CAN_E_PARAM_CONTROLLER and return E_NOT_OK.⌋*()*

[SWS_Can_00200] ⌈If development error detection for the Can module is enabled: if an invalid transition has been requested, the function Can_SetControllerMode shall raise the error CAN_E_TRANSITION and return E_NOT_OK.⌋*()*

[SWS_Can_00603] ⌈If selective wakeup is supported by hardware and the requested mode is CAN_CS_STARTED,CAN controller shall call the API CanIf_ConfirmCtrlPnAvailability() for the corresponding abstract CanIf ControllerId. CanIf_ConfirmCtrlPnAvailability informs CanNm (through CanIf and CanSm) that selective wakeup is enabled.⌋*()*

8.3.2.3 Can_DisableControllerInterrupts

[SWS_Can_00231] Definition of API function Can_DisableControllerInterrupts ⌈

Service Name	Can_DisableControllerInterrupts	

Syntax	`void Can_DisableControllerInterrupts (`
	`  uint8 Controller`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Controller	CAN controller for which interrupts shall be disabled.
Parameters (inout)	None	

▽

△

Parameters (out)	None
Return value	None
Description	This function disables all interrupts for this CAN controller.
Available via	Can.h

⌋*(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] Definition of API function Can_EnableControllerInterrupts ⌈

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.	
Available via	Can.h	

⌋*(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".

[SWS_Can_00209] ⌈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.⌋*()*

[SWS_Can_00210] ⌈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

[SWS_Can_00360] Definition of API function Can_CheckWakeup ⌈

Service Name	Can_CheckWakeup	
Syntax	`Std_ReturnType Can_CheckWakeup ( uint8 Controller )`	
Service ID [hex]	0x0b	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Controller	Controller to be checked for a wakeup.

▽

△

Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: API call has been accepted `E_NOT_OK`: API call has not been accepted
Description	This function checks if a wakeup has occurred for the given controller.	
Available via	Can.h	

⌋*()*

[SWS_Can_00361] ⌈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.⌋*()*

[SWS_Can_00362] ⌈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.⌋*()*

[SWS_Can_00363] ⌈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

[SWS_Can_91004] Definition of API function Can_GetControllerErrorState ⌈

Service Name	Can_GetControllerErrorState	

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

Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same ControllerId	
Parameters (in)	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState.
Parameters (inout)	None	
Parameters (out)	ErrorStatePtr	Pointer to a memory location, where the error state of the CAN controller will be stored.
Return value	Std_ReturnType	`E_OK`: Error state request has been accepted. `E_NOT_OK`: Error state request has not been accepted.
Description	This service obtains the error state of the CAN controller.	
Available via	Can.h	

⌋*()*

[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)*

8.3.2.7 Can_GetControllerMode

[SWS_Can_91014] Definition of API function Can_GetControllerMode ⌈

Service Name	Can_GetControllerMode	

Syntax	`Std_ReturnType Can_GetControllerMode (`
	`  uint8 Controller,`
	`  Can_ControllerStateType* ControllerModePtr`
	`)`	

Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Controller	CAN controller for which the status shall be requested.
Parameters (inout)	None	
Parameters (out)	ControllerModePtr	Pointer to a memory location, where the current mode of the CAN controller will be stored.
Return value	Std_ReturnType	E_OK: Controller mode request has been accepted. E_NOT_OK: Controller mode request has not been accepted.
Description	This service reports about the current status of the requested CAN controller.	
Available via	Can.h	

⌋*()*

[SWS_Can_91015] ⌈The service Can_GetControllerMode shall return the mode of the requested CAN controller.⌋*()*

[SWS_Can_91016] ⌈If development error detection for the Can module is enabled: The function Can_GetControllerMode shall raise the error CAN_E_UNINIT and return E_NOT_OK if the driver is not yet initialized.⌋*(SRS_BSW_00406, SRS_BSW_00416)*

[SWS_Can_91017] ⌈If parameter Controller of Can_GetControllerMode() has an invalid value, the CanDrv shall report development error code CAN_E_PARAM_CONTROLLER to the Det_ReportError service of the DET.⌋*(SRS_BSW_00323)*

[SWS_Can_91018] ⌈If parameter ControllerModePtr of `Can_GetControllerMode`() has an null pointer, the CanDrv shall report development error code `CAN_E_PARAM_POINTER` to the Det_ReportError service of the DET.⌋*(SRS_BSW_00323)*

8.3.2.8 canGetControllerRxErrorCounter

[SWS_Can_00511] Definition of API function Can_GetControllerRxErrorCounter
⌈

Service Name	Can_GetControllerRxErrorCounter	
Syntax	`Std_ReturnType Can_GetControllerRxErrorCounter ( uint8 ControllerId, uint8* RxErrorCounterPtr )`	
Service ID [hex]	0x30	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same ControllerId	
Parameters (in)	ControllerId	CAN controller, whose current Rx error counter shall be acquired.
Parameters (inout)	None	
Parameters (out)	RxErrorCounterPtr	Pointer to a memory location, where the current Rx error counter of the CAN controller will be stored.
Return value	Std_ReturnType	`E_OK`: Rx error counter available. `E_NOT_OK`: Wrong ControllerId, or Rx error counter not available.
Description	Returns the Rx error counter for a CAN controller. This value might not be available for all CAN controllers, in which case E_NOT_OK would be returned. Please note that the value of the counter might not be correct at the moment the API returns it, because the Rx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state.	
Available via	Can.h	

⌋*()*

[SWS_Can_00512] ⌈If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can_GetControllerRxErrorCounter shall raise development error `CAN_E_UNINIT` and return E_NOT_OK.⌋*(SRS_BSW_00406)*

[SWS_Can_00513] ⌈If development error detection for the Can module is enabled: if the parameter ControllerId is out of range, the function Can_GetControllerRxErrorCounter shall raise development error `CAN_E_PARAM_CONTROLLER` and return E_NOT_OK.⌋*(SRS_BSW_00323)*

[SWS_Can_00514] ⌈If development error detection for the Can module is enabled: if the parameter RxErrorCounterPtr is a null pointer, the function Can_GetControllerRxErrorCounter shall raise development error `CAN_E_PARAM_POINTER` and return E_NOT_OK.⌋*(SRS_BSW_00323)*

[SWS_Can_00515] ⌈When the API Can_GetControllerRxErrorCounter is called with Controller Id as input parameter then Can driver shall read the Rx error counter register of Can Controller and shall return the Rx error count to upper layer.⌋*(SRS_Can_01170)*

8.3.2.9 Can_GetControllerTxErrorCounter

[SWS_Can_00516] Definition of API function Can_GetControllerTxErrorCounter
⌈

Service Name	Can_GetControllerTxErrorCounter	

Syntax	`Std_ReturnType Can_GetControllerTxErrorCounter (`
	`  uint8 ControllerId,`
	`  uint8* TxErrorCounterPtr`
	`)`	

Service ID [hex]	0x31	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same ControllerId	
Parameters (in)	ControllerId	CAN controller, whose current Tx error counter shall be acquired.
Parameters (inout)	None	
Parameters (out)	TxErrorCounterPtr	Pointer to a memory location, where the current Tx error counter of the CAN controller will be stored.
Return value	Std_ReturnType	`E_OK`: Tx error counter available. `E_NOT_OK`: Wrong ControllerId, or Tx error counter not available.
Description	Returns the Tx error counter for a CAN controller. This value might not be available for all CAN controllers, in which case E_NOT_OK would be returned. Please note that the value of the counter might not be correct at the moment the API returns it, because the Tx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state.	
Available via	Can.h	

⌋*()*

[SWS_Can_00517] ⌈If development error detection for the Can module is enabled: if the module is not yet initialized, the function `Can_GetControllerTxErrorCounter` shall raise development error `CAN_E_UNINIT` and return E_NOT_OK.⌋*(SRS_BSW_00406)*

[SWS_Can_00518] ⌈If development error detection for the Can module is enabled: if the parameter ControllerId is out of range, the function `Can_GetControllerTxErrorCounter` shall raise development error `CAN_E_PARAM_CONTROLLER` and return E_NOT_OK.⌋*(SRS_BSW_00323)*

[SWS_Can_00519] ⌈If development error detection for the Can module is enabled: if the parameter TxErrorCounterPtr is a null pointer, the function `Can_GetControllerTxErrorCounter` shall raise development error `CAN_E_PARAM_POINTER` and return E_NOT_OK.⌋*(SRS_BSW_00323)*

[SWS_Can_00520] ⌈When the API `Can_GetControllerTxErrorCounter` is called with Controller Id as input parameter then Can driver shall read the Tx error counter register of Can Controller and shall return the Tx error count to upper layer.⌋*(SRS_Can_01170)*

8.3.2.10 Can_GetCurrentTime

[SWS_CAN_91026]{DRAFT} **Definition of API function Can_GetCurrentTime** ⌈

Service Name	Can_GetCurrentTime (draft)	
Syntax	`Std_ReturnType Can_GetCurrentTime ( uint8 ControllerId, Can_TimeStampType* timeStampPtr )`	
Service ID [hex]	0x32	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ControllerId	Index of the addresses CAN controller.
Parameters (inout)	None	
Parameters (out)	timeStampPtr	current time stamp
Return value	Std_ReturnType	`E_OK`: successful `E_NOT_OK`: failed
Description	Returns a time value out of the HW registers according to the capability of the HW Important Note: Can_GetCurrentTime may be called within an exclusive area. **Tags:** atp.Status=draft	
Available via	Can.h	

⌋*(SRS_Can_01181)*

[SWS_Can_00521]{DRAFT} ⌈If development error detection is enabled: the function shall check that the service `Can_Init` was previously called. If the check fails, the function shall raise the development error `CAN_E_UNINIT`.⌋*()*

[SWS_Can_00522]{DRAFT} ⌈If development error detection is enabled: the function shall check the parameter ControllerId for being valid. If the check fails, the function shall raise the development error `CAN_E_PARAM_CONTROLLER`.⌋*()*

[SWS_Can_00523]{DRAFT} ⌈If development error detection is enabled: the function shall check the parameter timeStampPtr for being valid. If the check fails, the function shall raise the development error `CAN_E_PARAM_POINTER`.⌋*()*

[SWS_Can_00524]{DRAFT} ⌈The function shall be pre-compile time configurable On/Off by the configuration parameter: CanGlobalTimeSupport.⌋*()*

8.3.2.11 Can_EnableEgressTimeStamp

[SWS_CAN_91025]{DRAFT} **Definition of API function Can_EnableEgressTime Stamp** ⌈

Service Name	Can_EnableEgressTimeStamp (draft)
Syntax	`void Can_EnableEgressTimeStamp ( Can_HwHandleType Hth )`

▽

△

Service ID [hex]	0x33	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Hth	information which HW-transmit handle shall be used for enabling the time stamp. Note: This is the smallest granularity which can be added for enabling the timestamp, at HTH level, without affecting the performance.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Activates egress time stamping on a dedicated HTH. Some HW does store once the egress time stamp marker and some HW needs it always before transmission. There will be no "disable" functionality, due to the fact, that the message type is always "time stamped" by network design. **Tags:** atp.Status=draft	
Available via	Can.h	

⌋*(SRS_Can_01181)*

[SWS_Can_00525]{DRAFT} ⌈If development error detection is enabled: the function shall check that the service `Can_Init` was previously called. If the check fails, the function shall raise the development error `CAN_E_UNINIT`.⌋*()*

[SWS_Can_00526]{DRAFT} ⌈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_00527]{DRAFT} ⌈The function shall be pre compile time configurable On/Off by the configuration parameter: CanGlobalTimeSupport.⌋*()*

[SWS_Can_00528]{DRAFT} ⌈Caveat: The function requires previous controller initialization (`Can_Init`).⌋*()*

8.3.2.12 Can_GetEgressTimeStamp

[SWS_CAN_91027]{DRAFT} **Definition of API function Can_GetEgressTimeStamp** ⌈

Service Name	Can_GetEgressTimeStamp (draft)

Syntax	`Std_ReturnType Can_GetEgressTimeStamp (`
	`  PduIdType TxPduId,`
	`  Can_HwHandleType Hth,`
	`  Can_TimeStampType* timeStampPtr`
	`)`

Service ID [hex]	0x34
Sync/Async	Synchronous

▽

△

Reentrancy	Non Reentrant for the same TxPduId.	
Parameters (in)	TxPduId	L-PDU handle of CAN L-PDU for which the time stamp shall be returned.
	Hth	HW-transmit handle for which the egress timestamp shall be retrieved
Parameters (inout)	None	
Parameters (out)	timeStampPtr	current time stamp
Return value	Std_ReturnType	E_OK: success E_NOT_OK: failed to read time stamp.
Description	Reads back the egress time stamp on a dedicated message object. It needs to be called within the TxConfirmation() function. **Tags:** atp.Status=draft	
Available via	Can.h	

⌋*(SRS_Can_01181)*

[SWS_Can_00529]{DRAFT} ⌈If development error detection is enabled: the function shall check that the service Can_Init was previously called. If the check fails, the function shall raise the development error CAN_E_UNINIT.⌋*()*

[SWS_Can_00530]{DRAFT} ⌈If development error detection is enabled: the function shall check the parameter TxPduId for being valid. If the check fails, the function shall raise the development error CAN_E_PARAM_LPDU.⌋*()*

[SWS_Can_00531]{DRAFT} ⌈If development error detection for the Can module is enabled: The function Can_GetEgressTimeStamp 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_00532]{DRAFT} ⌈If development error detection is enabled: the function shall check the parameter timeStampPtr for being valid. If the check fails, the function shall raise the development error CAN_E_PARAM_POINTER.⌋*()*

[SWS_Can_00533]{DRAFT} ⌈The function shall be pre-compile time configurable On/Off by the configuration parameter: CanGlobalTimeSupport.⌋*()*

[SWS_Can_00534]{DRAFT} ⌈Caveat: The function requires previous controller initialization (Can_Init).⌋*()*

8.3.2.13 Can_GetIngressTimeStamp

[SWS_CAN_91028]{DRAFT} **Definition of API function Can_GetIngressTimeStamp** ⌈

Service Name	Can_GetIngressTimeStamp (draft)	
Syntax	`Std_ReturnType Can_GetIngressTimeStamp ( Can_HwHandleType Hrh, Can_TimeStampType* timeStampPtr )`	
Service ID [hex]	0x35	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant for the same Hrh, Reentrant for different Hrh	
Parameters (in)	Hrh	HW-receive handle for which the ingress timestamp shall be retrieved
Parameters (inout)	None	
Parameters (out)	timeStampPtr	current time stamp
Return value	Std_ReturnType	`E_OK`: success `E_NOT_OK`: failed to read time stamp.
Description	Reads back the ingress time stamp on a dedicated message object. It needs to be called within the RxIndication() function. **Tags:** atp.Status=draft	
Available via	Can.h	

⌋*(SRS_Can_01181)*

[SWS_Can_00535]{DRAFT} ⌈If development error detection is enabled: the function shall check that the service `Can_Init` was previously called. If the check fails, the function shall raise the development error `CAN_E_UNINIT`.⌋*()*

[SWS_Can_00536]{DRAFT} ⌈If development error detection for the Can module is enabled: The function Can_GetIngressTimeStamp shall raise the error `CAN_E_PARAM_HANDLE` and shall return E_NOT_OK if the parameter `Hrh` is not a configured Hardware Receive Handle.⌋*()*

[SWS_Can_00537]{DRAFT} ⌈If development error detection is enabled: the function shall check the parameter timeStampPtr for being valid. If the check fails, the function shall raise the development error `CAN_E_PARAM_POINTER`.⌋*()*

[SWS_Can_00538]{DRAFT} ⌈The function shall be pre-compile time configurable On/Off by the configuration parameter: CanGlobalTimeSupport.⌋*()*

[SWS_Can_00539]{DRAFT} ⌈Caveat: The function requires previous controller initialization (`Can_Init`).⌋*()*

8.3.3 Services affecting a Hardware Handle

8.3.3.1 Can_Write

[SWS_Can_00233] Definition of API function Can_Write ⌈

Service Name	Can_Write	
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. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside one hardware unit.
	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 Can_ReturnType)
Description	This function is called by CanIf to pass a CAN message to CanDrv for transmission.	
Available via	Can.h	

⌋*(SRS_BSW_00312)*

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 to Chapter 8.2.3).

⌋*(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] Definition of configurable interface <LPDU_CalloutName> ⌈

Service Name	<LPDU_CalloutName>	

Syntax	`boolean <LPDU_CalloutName> (`
	`  uint8 Hrh,`
	`  Can_IdType CanId,`
	`  uint8 CanDataLegth,`
	`  const uint8* CanSduPtr`
	`)`	

Service ID [hex]	0x20	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Hrh	–
	CanId	–
	CanDataLegth	–
	CanSduPtr	–

▽

△

Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	–
Description	–	
Available via	Can_Externals.h	

⌋*()* where <LPDU_CalloutName> has to be substituted with the concrete L-PDU callout name which is configurable, see [ECUC_Can_00434].

[SWS_Can_00444] ⌈If the L-PDU callout returns false, the L-PDU shall not be processed any further.⌋*()*

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

8.5.1.1 Can_MainFunction_Write

[SWS_Can_00225] Definition of scheduled function Can_MainFunction_Write ⌈

Service Name	Can_MainFunction_Write

Syntax	`void Can_MainFunction_Write (`
	`  void`
	`)`

Service ID [hex]	0x01
Description	This function performs the polling of TX confirmation when CAN_TX_PROCESSING is set to POLLING.
Available via	SchM_Can.h

⌋*()*

[SWS_Can_00031] ⌈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.⌋*(SRS_BSW_00432, SRS_BSW_00373, SRS_SPAL_00157)*

[SWS_Can_00178] ⌈The Can module may implement the function `Can_MainFunction_Write` as empty define in case no polling at all is used.⌋*()*

[SWS_Can_00441] ⌈If more than one main function period is configured by CanMainFunctionRWPeriods (see `ECUC_Can_00437`), the name of the `Can_MainFunction_Write`() functions shall be

- `Can_MainFunction_Write`_<CanMainFunctionRWPeriods.ShortName>() for each CanMainFunctionRWPeriods that is referenced by at least one TRANSMIT CanHardwareObject (see `ECUC_Can_00438`).

⌋*()*

8.5.1.2 Can_MainFunction_Read

[SWS_Can_00226] Definition of scheduled function Can_MainFunction_Read ⌈

Service Name	Can_MainFunction_Read

Syntax	`void Can_MainFunction_Read (`
	`  void`
	`)`

Service ID [hex]	0x08
Description	This function performs the polling of RX indications when CAN_RX_PROCESSING is set to POLLING.
Available via	SchM_Can.h

⌋*()*

[SWS_Can_00108] ⌈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.⌋*(SRS_BSW_00432, SRS_SPAL_00157)*

[SWS_Can_00180] ⌈The Can module may implement the function Can_MainFunction_Read as empty define in case no polling at all is used.⌋*()*

[SWS_Can_00442] ⌈If more than one main function period is configured by CanMainFunctionRWPeriods (see ECUC_Can_00437), the name of the Can_MainFunction_Read() functions shall be

- Can_MainFunction_Read_<CanMainFunctionRWPeriods.ShortName>() for each CanMainFunctionRWPeriods that is referenced by at least one RECEIVE CanHardwareObject (see ECUC_Can_00438).

⌋*()*

8.5.1.3 Can_MainFunction_BusOff

[SWS_Can_00227] Definition of scheduled function Can_MainFunction_BusOff ⌈

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'.
Available via	SchM_Can.h

⌋*()*

[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] Definition of scheduled function Can_MainFunction_Wakeup ⌈

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'.
Available via	SchM_Can.h

⌋*()*

[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] Definition of scheduled function Can_MainFunction_Mode ⌈

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.
Available via	SchM_Can.h

⌋*()*

[SWS_Can_00369] ⌈The function `Can_MainFunction_Wakeup` 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.

[SWS_Can_00234] Definition of mandatory interfaces in module Can ⌈

API Function	***Header File***	***Description***
CanIf_ControllerBusOff	CanIf_Can.h	This service indicates a Controller BusOff event referring to the corresponding CAN Controller with the abstract CanIf ControllerId.
CanIf_ControllerModeIndication	CanIf_Can.h	This service indicates a controller state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId.
CanIf_RxIndication	CanIf_Can.h	This service indicates a successful reception of a received CAN Rx L-PDU to the CanIf after passing all filters and validation checks.
CanIf_TxConfirmation	CanIf_Can.h	This service confirms a previously successfully processed transmission of a CAN TxPDU.
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
GetCounterValue	Os.h	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).

⌋*(SRS_Can_01055)*

8.6.2 Optional Interfaces

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

[SWS_Can_00235] Definition of optional interfaces in module Can ⌈

API Function	***Header File***	***Description***
CanIf_ConfirmCtrlPnAvailability (draft)	CanIf_CanDrv.h	This service indicates that the controller is running in PN communication mode referring to the corresponding CAN controller with the abstract Can If ControllerId. **Tags:** atp.Status=draft
CanIf_ControllerErrorStatePassive	CanIf_Can.h	The function derives the ErrorCounterTreshold from RxErrorCounter/ TxErrorCounter values and reports it to the IdsM as security event SEV_CAN_ ERRORSTATE_PASSIVE to the IdsM. It also prepares the context data for the respective security event.

▽

△

API Function	*Header File*	*Description*
CanIf_ErrorNotification	CanIf_Can.h	The function shall derive the bus error source rx or tx from the parameter CanError and report the bus error as security event SEV_CAN_TX_ERROR_DETECTED or SEV_CAN_RX_ERROR_DETECTED. It also prepares the context data for the respective security event.
CanIf_TriggerTransmit	CanIf.h	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.
Det_ReportError	Det.h	Service to report development errors.
EcuM_CheckWakeup	EcuM.h	This function can be called to check the given wakeup sources. It will pass the argument to the integrator function EcuM_CheckWakeupHook. It can 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.
EcuM_SetWakeupEvent	EcuM.h	Sets the wakeup event.
Icu_DisableNotification	Icu.h	This function disables the notification of a channel.
Icu_EnableNotification	Icu.h	This function enables the notification on the given channel.

⌋*(SRS_SPAL_12056, SRS_Can_01054)*

8.6.3 Configurable Interfaces

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

9 Sequence diagrams

9.1 Interaction between Can and CanIf module

For sequence diagrams see the CanIf module Specification [1].
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 [5].

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 [9]

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)*

[SWS_Can_00507] ⌈The Can Driver module shall reject configurations with partition mappings which are not supported by the implementation.⌋*()*

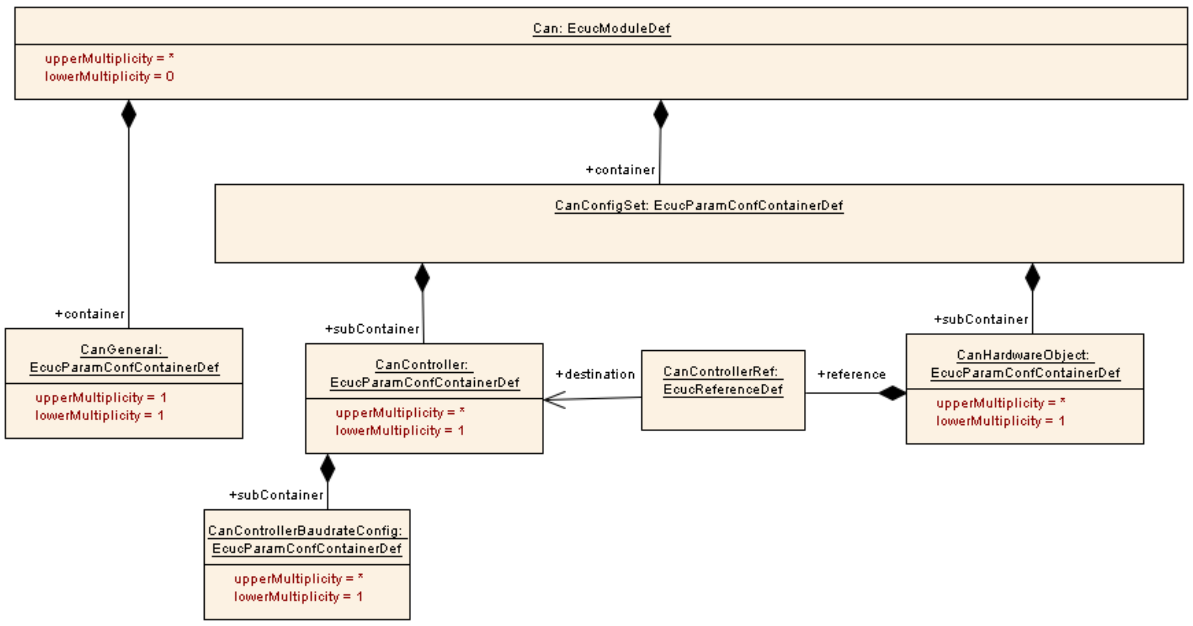

Figure 10.1: Can Module Configuration Layout

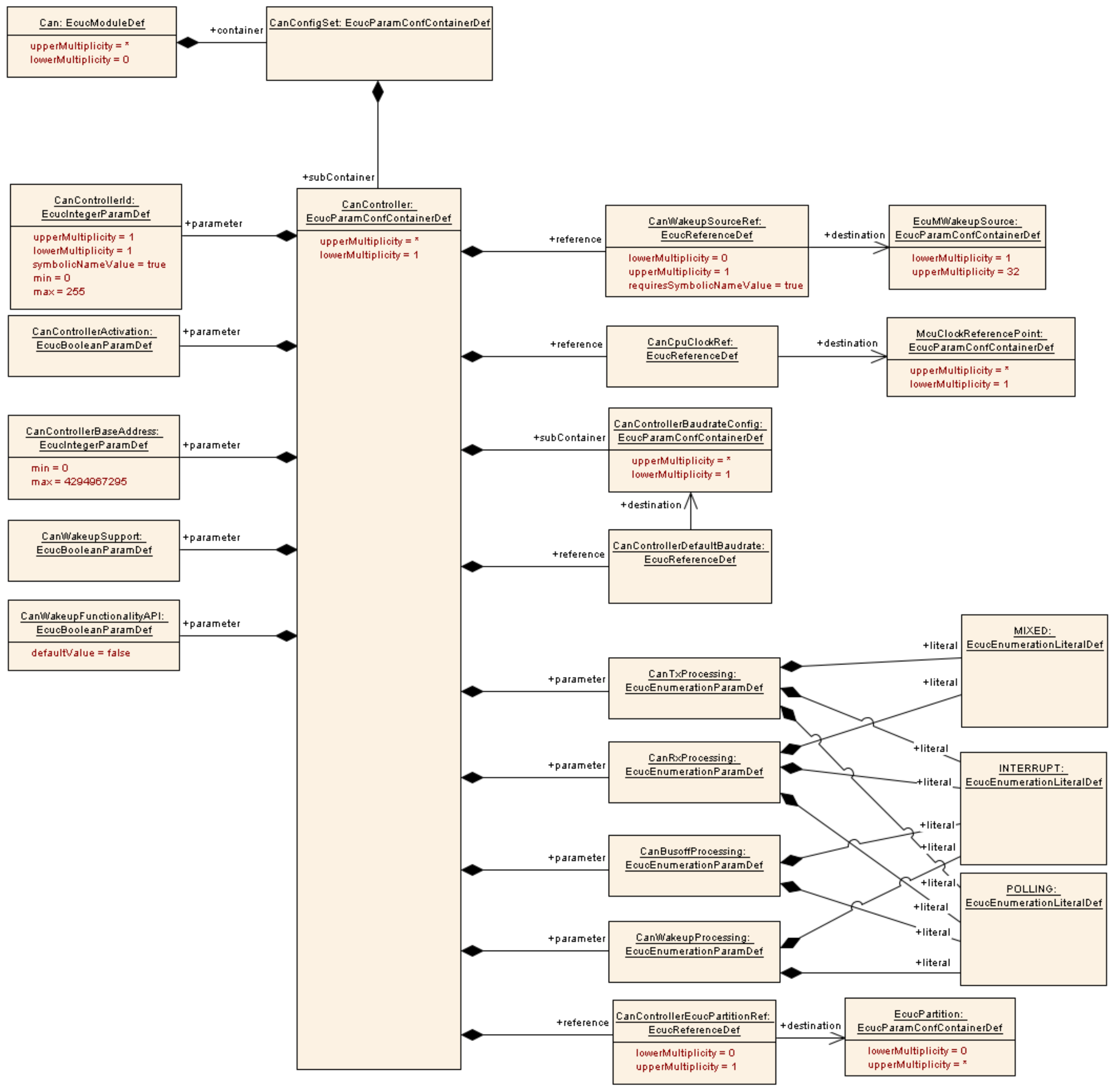
Can: EcucModuleDef
upperMultiplicity = *
lowerMultiplicity = 0
+container
CanConfigSet: EcucParamConfContainerDef
+subContainer
CanController: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
CanControllerId: EcucIntegerParamDef
upperMultiplicity = 1
lowerMultiplicity = 1
symbolicNameValue = true
min = 0
max = 255
+parameter
CanControllerActivation: EcucBooleanParamDef
CanControllerBaseAddress: EcucIntegerParamDef
min = 0
max = 4294967295
CanWakeupSupport: EcucBooleanParamDef
CanWakeupFunctionalityAPI: EcucBooleanParamDef
defaultValue = false
+reference
CanWakeupSourceRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
EcuMWakeupSource: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 32
CanCpuClockRef: EcucReferenceDef
McuClockReferencePoint: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
CanControllerBaudrateConfig: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
CanControllerDefaultBaudrate: EcucReferenceDef
CanTxProcessing: EcucEnumerationParamDef
CanRxProcessing: EcucEnumerationParamDef
CanBusoffProcessing: EcucEnumerationParamDef
CanWakeupProcessing: EcucEnumerationParamDef
+literal
MIXED: EcucEnumerationLiteralDef
INTERRUPT: EcucEnumerationLiteralDef
POLLING: EcucEnumerationLiteralDef
CanControllerEcucPartitionRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
EcucPartition: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *

Figure 10.2: Can Controller Configuration Layout

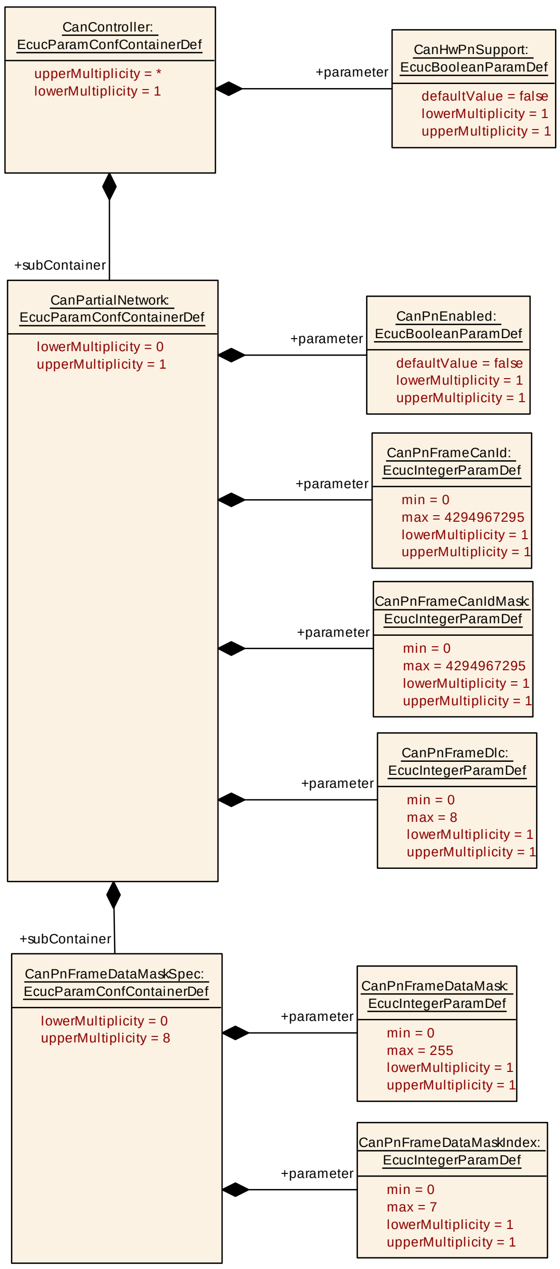

Figure 10.3: Can Controller Partial Network Configuration

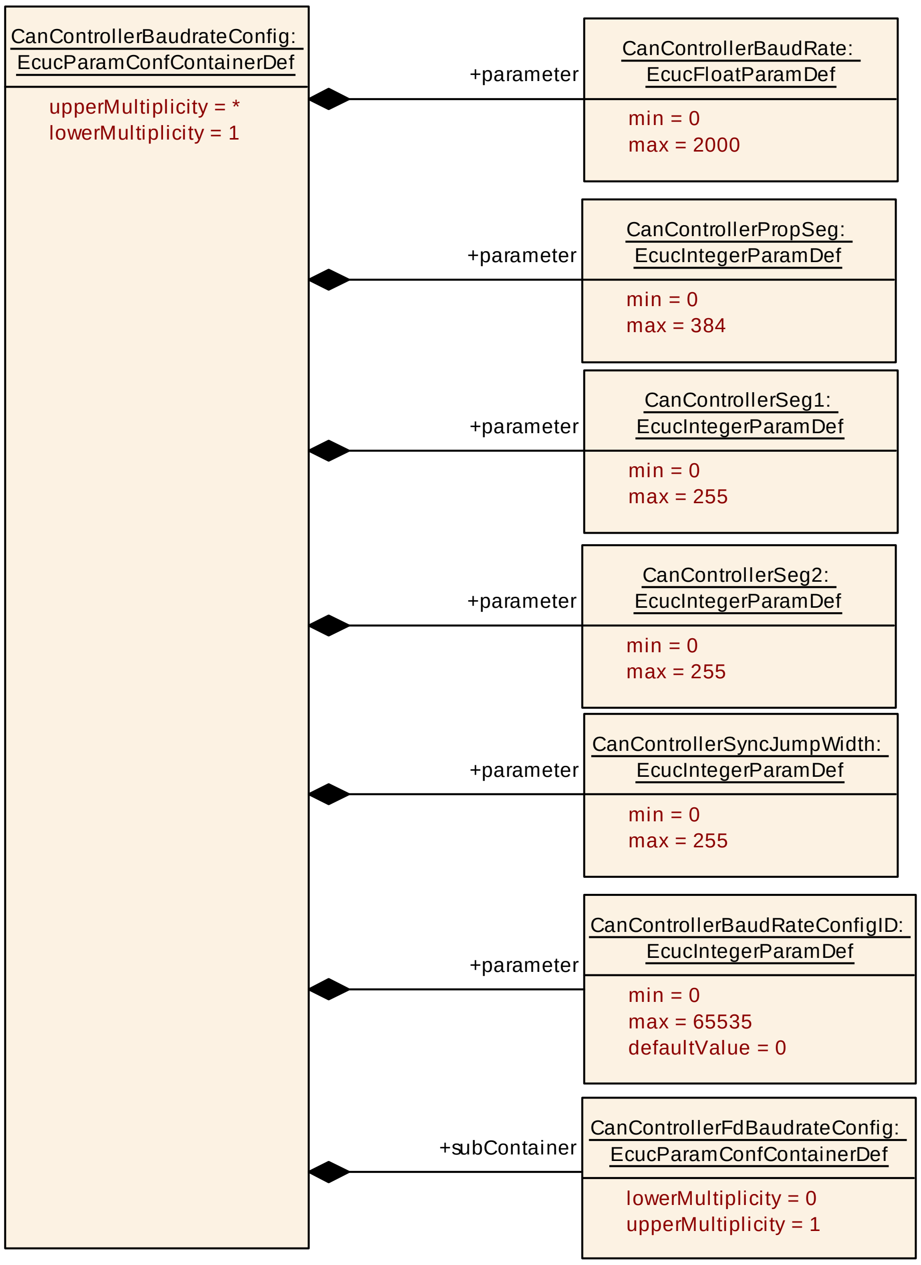
CanControllerBaudrateConfig: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+parameter
CanControllerBaudRate: EcucFloatParamDef
min = 0
max = 2000
+parameter
CanControllerPropSeg: EcucIntegerParamDef
min = 0
max = 384
+parameter
CanControllerSeg1: EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerSeg2: EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerSyncJumpWidth: EcucIntegerParamDef
min = 0
max = 255
+parameter
CanControllerBaudRateConfigID: EcucIntegerParamDef
min = 0
max = 65535
defaultValue = 0
+subContainer
CanControllerFdBaudrateConfig: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1

Figure 10.4: Can Controller Baud Rate Configuration Layout

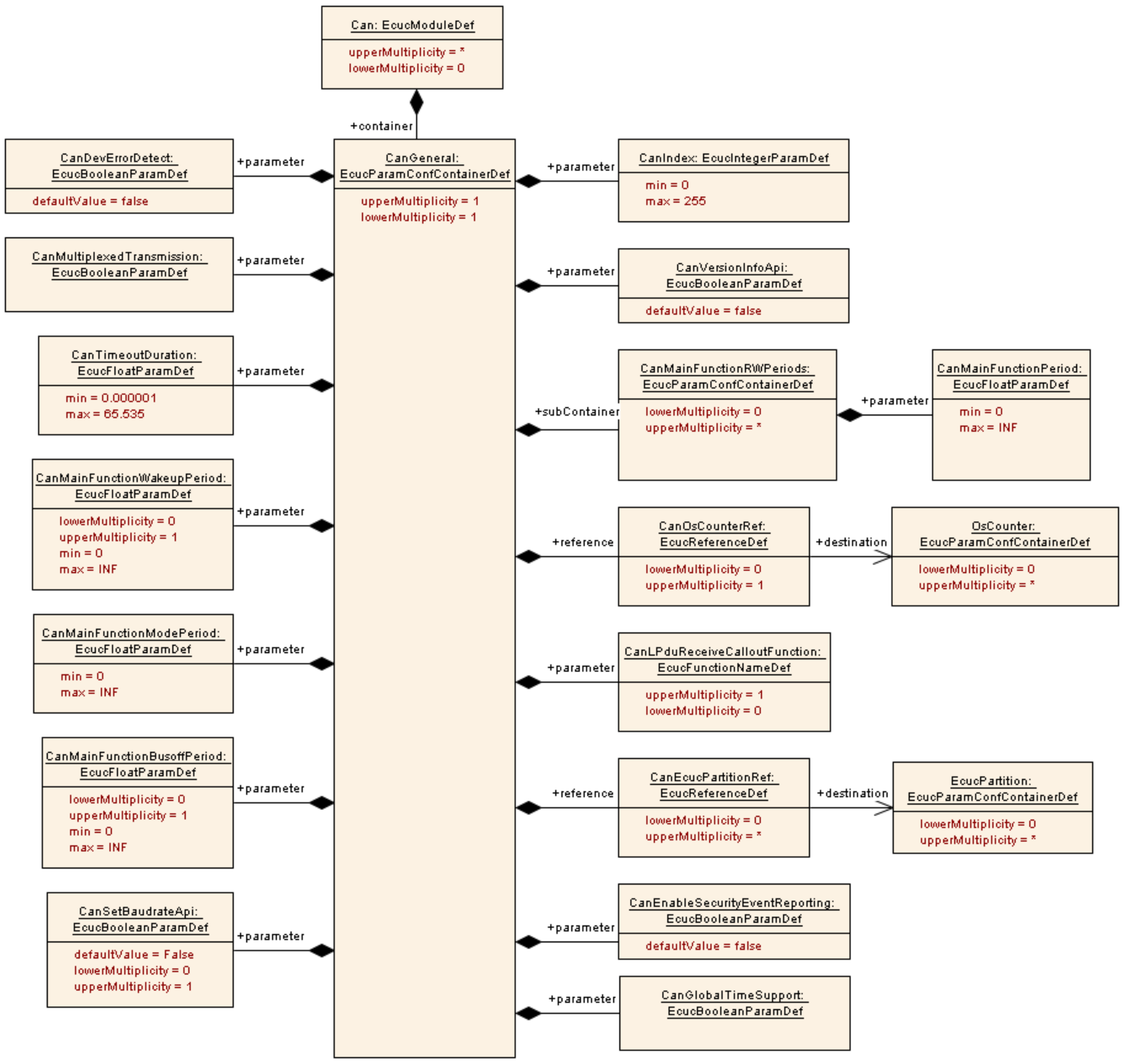
Can: EcucModuleDef
upperMultiplicity = *
lowerMultiplicity = 0
+container
CanGeneral: EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 1
+parameter
CanDevErrorDetect: EcucBooleanParamDef
defaultValue = false
CanMultiplexedTransmission: EcucBooleanParamDef
CanTimeoutDuration: EcucFloatParamDef
min = 0.000001
max = 65.535
CanMainFunctionWakeupPeriod: EcucFloatParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
min = 0
max = INF
CanMainFunctionModePeriod: EcucFloatParamDef
min = 0
max = INF
CanMainFunctionBusoffPeriod: EcucFloatParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
min = 0
max = INF
CanSetBaudrateApi: EcucBooleanParamDef
defaultValue = False
lowerMultiplicity = 0
upperMultiplicity = 1
CanIndex: EcucIntegerParamDef
min = 0
max = 255
CanVersionInfoApi: EcucBooleanParamDef
defaultValue = false
+subContainer
CanMainFunctionRWPeriods: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CanMainFunctionPeriod: EcucFloatParamDef
min = 0
max = INF
+reference
CanOsCounterRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
OsCounter: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CanLPduReceiveCalloutFunction: EcucFunctionNameDef
upperMultiplicity = 1
lowerMultiplicity = 0
CanEcucPartitionRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = *
EcucPartition: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CanEnableSecurityEventReporting: EcucBooleanParamDef
defaultValue = false
CanGlobalTimeSupport: EcucBooleanParamDef

Figure 10.5: Can General Configuration Layout

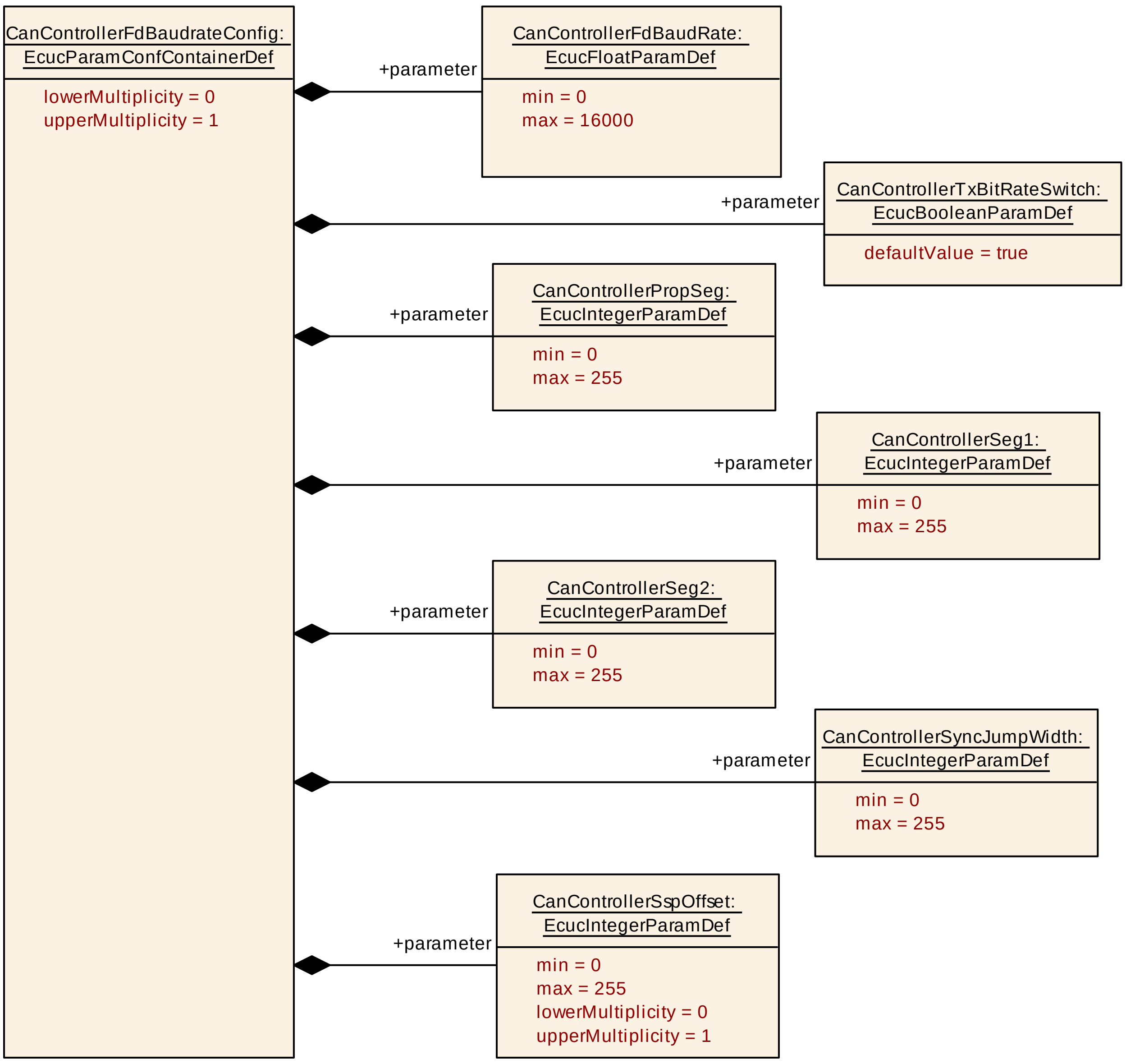
CanControllerFdBaudrateConfig: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
CanControllerFdBaudRate: EcucFloatParamDef
min = 0
max = 16000
+parameter
CanControllerTxBitRateSwitch: EcucBooleanParamDef
defaultValue = true
+parameter
CanControllerPropSeg: EcucIntegerParamDef
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
CanControllerSspOffset: EcucIntegerParamDef
min = 0
max = 255
lowerMultiplicity = 0
upperMultiplicity = 1

Figure 10.6: CanControllerFdBaudrateConfig

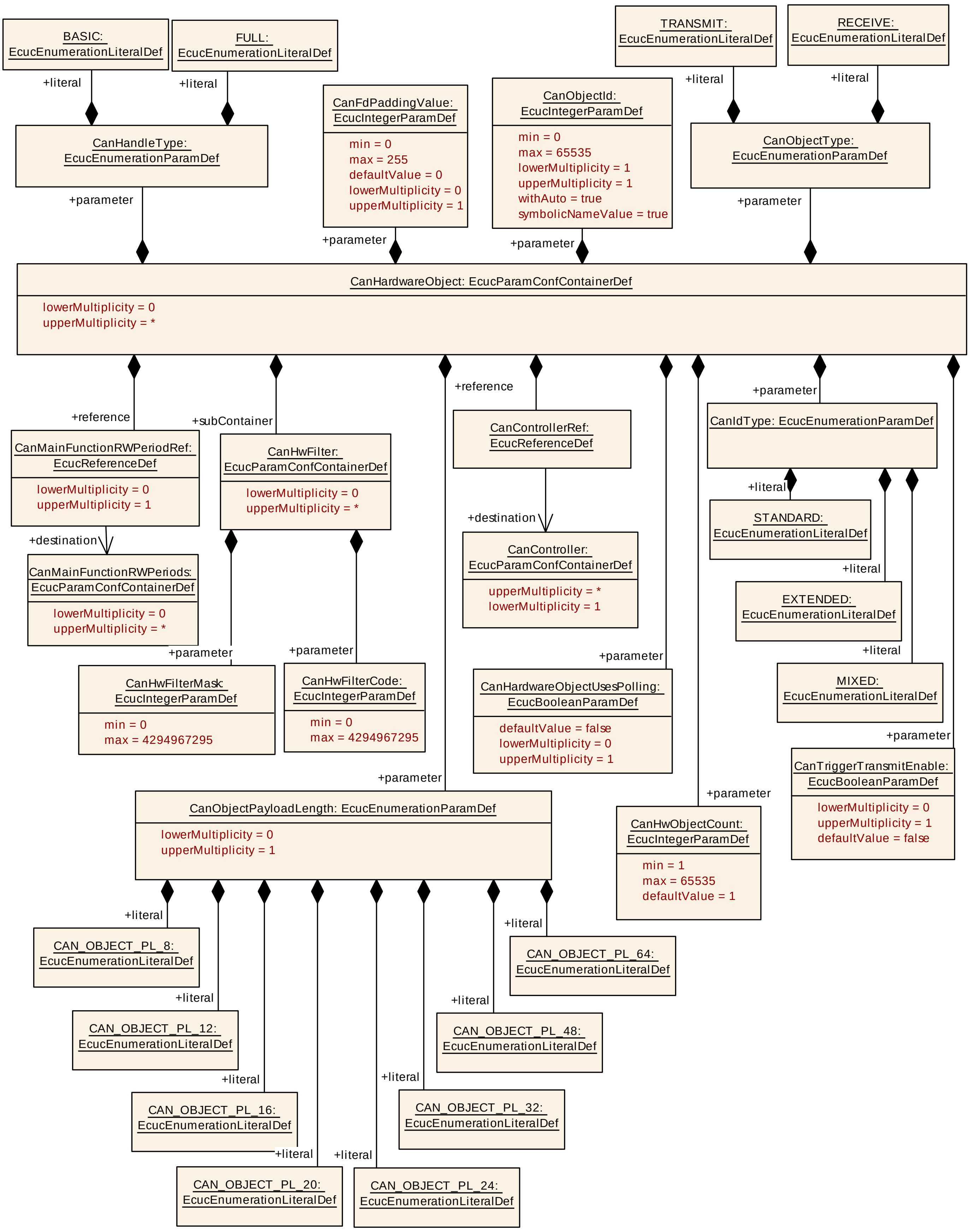
BASIC: EcucEnumerationLiteralDef
FULL: EcucEnumerationLiteralDef
+literal
+literal
CanHandleType: EcucEnumerationParamDef
+parameter
CanFdPaddingValue: EcucIntegerParamDef
min = 0
max = 255
defaultValue = 0
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
CanObjectId: EcucIntegerParamDef
min = 0
max = 65535
lowerMultiplicity = 1
upperMultiplicity = 1
withAuto = true
symbolicNameValue = true
+parameter
TRANSMIT: EcucEnumerationLiteralDef
RECEIVE: EcucEnumerationLiteralDef
+literal
+literal
CanObjectType: EcucEnumerationParamDef
+parameter
CanHardwareObject: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+reference
CanMainFunctionRWPeriodRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
CanMainFunctionRWPeriods: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
CanHwFilter: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
CanHwFilterMask: EcucIntegerParamDef
min = 0
max = 4294967295
+parameter
CanHwFilterCode: EcucIntegerParamDef
min = 0
max = 4294967295
+reference
CanControllerRef: EcucReferenceDef
+destination
CanController: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+parameter
CanHardwareObjectUsesPolling: EcucBooleanParamDef
defaultValue = false
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
CanIdType: EcucEnumerationParamDef
+literal
STANDARD: EcucEnumerationLiteralDef
+literal
EXTENDED: EcucEnumerationLiteralDef
+literal
MIXED: EcucEnumerationLiteralDef
+parameter
CanTriggerTransmitEnable: EcucBooleanParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
defaultValue = false
+parameter
CanHwObjectCount: EcucIntegerParamDef
min = 1
max = 65535
defaultValue = 1
+parameter
CanObjectPayloadLength: EcucEnumerationParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+literal
CAN_OBJECT_PL_8: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_12: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_16: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_20: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_24: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_32: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_48: EcucEnumerationLiteralDef
+literal
CAN_OBJECT_PL_64: EcucEnumerationLiteralDef

Figure 10.7: Can Hardware Object Configuration Layout

10.2.1 Can

SWS Item	[ECUC_Can_00489]
Module Name	Can
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		
Container Name	**Multiplicity**	**Scope / Dependency**
CanConfigSet	1	This container contains the configuration parameters and sub containers of the AUTOSAR Can module.
CanGeneral	1	This container contains the parameters related each CAN Driver Unit.

10.2.2 CanGeneral

SWS Item	[ECUC_Can_00497]
Container Name	CanGeneral
Parent Container	Can
Description	This container contains the parameters related each CAN Driver Unit.
Configuration Parameters	

SWS Item	[ECUC_Can_00064]		
Parameter 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_00496]
Parameter Name	CanEnableSecurityEventReporting
Parent Container	CanGeneral
Description	Switches the reporting of security events to the IdsM: - true: reporting is enabled. - false: reporting is disabled. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef

▽

△

Default value	false		
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_00498]**		
Parameter Name	CanGlobalTimeSupport		
Parent Container	CanGeneral		
Description	Enables/Disables the Global Time APIs used when hardware timestamping is supported by CAN controller. **Tags:** atp.Status=draft		
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_00320]**		
Parameter 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	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_Can_00434]**
Parameter 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	–
Regular Expression	–
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: local		

SWS Item	**[ECUC_Can_00355]**		
Parameter 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 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			

SWS Item	**[ECUC_Can_00376]**		
Parameter Name	CanMainFunctionModePeriod		
Parent Container	CanGeneral		
Description	This parameter describes the period for cyclic call to Can_MainFunction_Mode. Unit is seconds.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

SWS Item	**[ECUC_Can_00357]**
Parameter Name	CanMainFunctionWakeupPeriod
Parent Container	CanGeneral
Description	This parameter describes the period for cyclic call to Can_MainFunction_Wakeup. Unit is seconds.

▽

△

Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	]0 .. 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			

SWS Item	**[ECUC_Can_00095]**		
Parameter Name	CanMultiplexedTransmission		
Parent Container	CanGeneral		
Description	Specifies if multiplexed transmission shall be supported.ON or OFF		
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: ECU dependency: CAN Hardware Unit supports multiplexed transmission		

SWS Item	**[ECUC_Can_00482]**		
Parameter 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 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_00113]**		
Parameter 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	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00106]**		
Parameter 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	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00491]**		
Parameter Name	CanEcucPartitionRef		
Parent Container	CanGeneral		
Description	Maps the CAN driver to zero or multiple ECUC partitions to make the modules API available in this partition. The CAN driver will operate as an independent instance in each of the partitions.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
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_00431]**		
Parameter Name	CanOsCounterRef		
Parent Container	CanGeneral		
Description	This parameter contains a reference to the OsCounter, which is used by the CAN driver.		
Multiplicity	0..1		
Type	Reference to OsCounter		
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: local		

SWS Item	**[ECUC_Can_00430]**		
Parameter Name	CanSupportTTCANRef		
Parent Container	CanGeneral		
Description	The parameter refers to CanIfSupportTTCAN parameter in the CAN Interface Module configuration. The CanIfSupportTTCAN parameter defines whether TTCAN is supported.		
Multiplicity	1		
Type	Reference to CanIfPrivateCfg		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanMainFunctionRWPeriods	0..*	This container contains the parameter for configuring the period for cyclic call to Can_MainFunction_Read or Can_Main Function_Write depending on the referring item.
CanXLGeneral	0..1	This container is specified in the SWS CAN XL Driver and contains global parameters of the CAN XL Driver.

[SWS_Can_CONSTR_00508] ⌈The module will operate as an independent instance in each of the partitions, means the called API will only target the partition it is called in.⌋ *()*

10.2.3 CanController

SWS Item	**[ECUC_Can_00354]**		
Container Name	CanController		
Parent Container	CanConfigSet		
Description	This container contains the configuration parameters of the CAN controller(s).		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_Can_00314]**		
Parameter Name	CanBusoffProcessing		
Parent Container	CanController		
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.	
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_00315]**		
Parameter 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]**	
Parameter Name	CanControllerBaseAddress	
Parent Container	CanController	
Description	Specifies the CAN controller base address.	
Multiplicity	1	
Type	EcucIntegerParamDef	
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]**		
Parameter 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	EcucIntegerParamDef (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 withAuto = true		

SWS Item	**[ECUC_Can_00529]**		
Parameter Name	CanHwPnSupport		
Parent Container	CanController		
Description	Indicates whether the HW supports the selective wakeup function. TRUE = Selective wakeup feature is supported by the CAN controller. FALSE = Selective wakeup functionality is not available in the CAN controller.		
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: CanWakeupSupport		

SWS Item	**[ECUC_Can_00317]**	
Parameter Name	CanRxProcessing	
Parent Container	CanController	
Description	Enables / disables API Can_MainFunction_Read() for handling PDU reception 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_00318]**		
Parameter 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_00319]**		
Parameter Name	CanWakeupProcessing		
Parent Container	CanController		
Description	Enables / disables API Can_MainFunction_Wakeup() for handling wakeup events in polling mode.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	INTERRUPT	Interrupt 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_00330]**		
Parameter Name	CanWakeupSupport		
Parent Container	CanController		
Description	CAN driver support for wakeup over CAN Bus.		
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			

SWS Item	**[ECUC_Can_00435]**		
Parameter 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**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00492]**		
Parameter Name	CanControllerEcucPartitionRef		
Parent Container	CanController		
Description	Maps the CAN controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the CAN driver is mapped to.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
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_00313]**		
Parameter 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**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

SWS Item	[ECUC_Can_00359]		
Parameter 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		
Multiplicity	0..1		
Type	Symbolic name reference to EcuMWakeupSource		
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: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanControllerBaudrateConfig	1..*	This container contains bit timing related configuration parameters of the CAN controller(s).
CanPartialNetwork	0..1	Container gives CAN Controller driver information about the configuration of Partial Networking functionality.
CanTTController	0..1	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.
CanXLController	0..1	This container is specified in the SWS CAN XL Driver and represents a CAN XL channel. If this container is present, the CAN driver will provide the extended CanXL API.

[SWS_Can_CONSTR_00509] ⌈The ECUC partitions referenced by CanControllerEcucPartitionRef shall be a subset of the ECUC partitions referenced by CanEcucPartitionRef.⌋*()*

[SWS_Can_CONSTR_00510] ⌈CanController and CanTrcvChannel of one communication channel shall all reference the same ECUC partition.⌋*()*

[SWS_Can_CONSTR_00511] ⌈If CanEcucPartitionRef references one or more ECUC partitions, CanControllerEcucPartitionRef shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋*()*

10.2.4 CanControllerBaudrateConfig

SWS Item	**[ECUC_Can_00387]**		
Container Name	CanControllerBaudrateConfig		
Parent Container	CanController		
Description	This container contains bit timing related configuration parameters of the CAN controller(s).		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**[ECUC_Can_00005]**		
Parameter Name	CanControllerBaudRate		
Parent Container	CanControllerBaudrateConfig		
Description	Specifies the baudrate of the controller in kbps.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 2000]		
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_00471]**		
Parameter Name	CanControllerBaudRateConfigID		
Parent Container	CanControllerBaudrateConfig		
Description	This ID is used by SetBaudrate API and uniquely identifies a specific baud rate configuration within a controller configuration.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	0		
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 dependency: CanSetBaudrateApi		

SWS Item	**[ECUC_Can_00073]**
Parameter Name	CanControllerPropSeg
Parent Container	CanControllerBaudrateConfig
Description	Specifies propagation delay in time quantas.

▽

△

Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 384		
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_00074]**		
Parameter Name	CanControllerSeg1		
Parent Container	CanControllerBaudrateConfig		
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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00075]**		
Parameter Name	CanControllerSeg2		
Parent Container	CanControllerBaudrateConfig		
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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00383]**		
Parameter 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		
Container Name	**Multiplicity**	**Scope / Dependency**
CanControllerFdBaudrateConfig	0..1	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.
CanXLBaudrateConfig	0..1	This container is specified in the SWS CAN XL Driver and contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN XL frame.

10.2.5 CanControllerFdBaudrateConfig

SWS Item	**[ECUC_Can_00473]**
Container Name	CanControllerFdBaudrateConfig
Parent Container	CanControllerBaudrateConfig
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]**		
Parameter Name	CanControllerFdBaudRate		
Parent Container	CanControllerFdBaudrateConfig		
Description	Specifies the data segment baud rate of the controller in kbps.		
Multiplicity	1		
Type	EcucFloatParamDef		
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]**
Parameter 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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00477]**		
Parameter 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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00478]**		
Parameter 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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00494]**
Parameter Name	CanControllerSspOffset
Parent Container	CanControllerFdBaudrateConfig

▽

△

Description	Specifies the Transmitter Delay Compensation Offset in minimum time quanta (see [17]). Transmitter Delay Compensation Offset is used to adjust the position of the Secondary Sample Point (SSP), relative to the beginning of the received bit. If this parameter is configured, the Transmitter Delay Compensation is done by measurement of the CAN controller. If not specified, Transmitter Delay Compensation is disabled. Note: MTQ == Minimum Time Quanta in seconds == 1/(frequency of the CAN controller clock) Secondary Sample Point Offset in seconds = CanControllerSspOffset * MTQ Example: CAN controller clock frequency = 20MHz => MTQ = 1/20 * 10ˆ(-6) s = 0,05 us = 50ns Baud rate = 1MBit/s => BitTime = 1/(1 * 10ˆ6) s/Bit = 1 * 10ˆ(-6) = 1us/Bit SSP = 75% => SSP in seconds = 0,75 * 1us = 750 ns CanControllerSspOffset in MTQ = 750ns / 50ns = 15 Note: Please consider the minimum range (0..63) stated in [17] and the range definition (0..127) used as per [19].		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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: local		

SWS Item	**[ECUC_Can_00479]**		
Parameter 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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00475]**
Parameter Name	CanControllerTxBitRateSwitch
Parent Container	CanControllerFdBaudrateConfig
Description	Specifies if the bit rate switching shall be used for transmissions. If FALSE: CAN FD frames shall be sent without bit rate switching.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	true

▽

△

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		

No Included Containers

10.2.6 CanPartialNetwork

SWS Item	**[ECUC_Can_00530]**		
Container Name	CanPartialNetwork		
Parent Container	CanController		
Description	Container gives CAN Controller driver information about the configuration of Partial Networking functionality.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_Can_00531]**		
Parameter Name	CanPnEnabled		
Parent Container	CanPartialNetwork		
Description	Indicates whether the selective wake-up function is enabled or disabled in HW. TRUE = Selective wakeup feature is enabled in the controller hardware FALSE = Selective wakeup feature is disabled in the controller hardware		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
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_00532]**	
Parameter Name	CanPnFrameCanId	
Parent Container	CanPartialNetwork	
Description	CAN ID of the Wake-up Frame (WUF).	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
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_00533]		
Parameter Name	CanPnFrameCanIdMask		
Parent Container	CanPartialNetwork		
Description	ID Mask for the selective activation of the CAN controller. It is used to enableFrame Wake-up (WUF) on a group of IDs.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
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_00535]		
Parameter Name	CanPnFrameDlc		
Parent Container	CanPartialNetwork		
Description	Data Length of the Wake-up Frame (WUF).		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 8		
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		
Container Name	Multiplicity	Scope / Dependency
CanPnFrameDataMaskSpec	0..8	Defines data payload mask to be used on the received payload in order to determine if the controller must be woken up by the received Wake-up Frame (WUF).

10.2.7 CanPnFrameDataMaskSpec

SWS Item	[ECUC_Can_00534]
Container Name	CanPnFrameDataMaskSpec
Parent Container	CanPartialNetwork

▽

△

Description	Defines data payload mask to be used on the received payload in order to determine if the controller must be woken up by the received Wake-up Frame (WUF).		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_Can_00536]**		
Parameter Name	CanPnFrameDataMask		
Parent Container	CanPnFrameDataMaskSpec		
Description	Defines the n byte (Byte0 = LSB) of the data payload mask to be used on the received payload in order to determine if the controller must be woken up by the received Wake-up Frame (WUF).		
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		

SWS Item	**[ECUC_Can_00537]**		
Parameter Name	CanPnFrameDataMaskIndex		
Parent Container	CanPnFrameDataMaskSpec		
Description	Holds the position n in frame of the mask-part		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
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		

No Included Containers

10.2.8 CanHardwareObject

SWS Item	**[ECUC_Can_00324]**
Container Name	CanHardwareObject
Parent Container	CanConfigSet

▽

△

Description	This container contains the configuration (parameters) of CAN Hardware Objects.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**[ECUC_Can_00485]**		
Parameter Name	CanFdPaddingValue		
Parent Container	CanHardwareObject		
Description	Specifies the value which is used to pad unspecified data in CAN FD frames > 8 bytes for transmission. This is necessary due to the discrete possible values of the DLC if > 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.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	0		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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_00323]**		
Parameter Name	CanHandleType		
Parent Container	CanHardwareObject		
Description	Specifies the type (Full-CAN or Basic-CAN) of a hardware object.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	BASIC	For several L-PDUs are hadled by the hardware object	
	FULL	For only one L-PDU (identifier) is handled by the hardware object	
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 dependency: This configuration element is used as information for the CAN Interface only. The relevant CAN driver configuration is done with the filter mask and identifier.		

SWS Item	**[ECUC_Can_00490]**
Parameter Name	CanHardwareObjectUsesPolling
Parent Container	CanHardwareObject
Description	Enables polling of this hardware object.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default value	false
Scope / Dependency	dependency: This parameter shall exist if CanRxProcessing/CanTxProcessing is set to Mixed.

SWS Item	**[ECUC_Can_00467]**		
Parameter 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	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	1		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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]**		
Parameter Name	CanIdType		
Parent Container	CanHardwareObject		
Description	Specifies whether the IdValue is of type standard identifier, extended identifier or 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 (11bit).	
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]**		
Parameter 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 withAuto = true		

SWS Item	**[ECUC_Can_00495]**		
Parameter Name	CanObjectPayloadLength		
Parent Container	CanHardwareObject		
Description	Specifies the maximum L-PDU payload length in bytes the hardware object can store. If the parameter is not provided, Can driver configuration generators have to assume the maximum length of the underlying CAN derivate, e.g. 8 bytes for CAN, 64 bytes for CAN-FD.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CAN_OBJECT_PL_12	Payload length of 12 Bytes	
	CAN_OBJECT_PL_16	Payload length of 16 Bytes	
	CAN_OBJECT_PL_20	Payload length of 20 Bytes	
	CAN_OBJECT_PL_24	Payload length of 24 Bytes	
	CAN_OBJECT_PL_32	Payload length of 32 Bytes	
	CAN_OBJECT_PL_48	Payload length of 48 Bytes	
	CAN_OBJECT_PL_64	Payload length of 64 Bytes	
	CAN_OBJECT_PL_8	Payload length of 8 Bytes	
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_00327]**
Parameter Name	CanObjectType
Parent Container	CanHardwareObject
Description	Specifies if the HardwareObject is used as Transmit or as Receive object
Multiplicity	1
Type	EcucEnumerationParamDef

▽

△

Range	RECEIVE	Receive HOH	
	TRANSMIT	Transmit HOH	
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_00486]**		
Parameter 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**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

SWS Item	**[ECUC_Can_00322]**		
Parameter 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**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: The referenced CanController has to contain a CanXLController.		

SWS Item	**[ECUC_Can_00438]**		
Parameter Name	CanMainFunctionRWPeriodRef		
Parent Container	CanHardwareObject		
Description	Reference to CanMainFunctionPeriod. If configured, this hardware object will be polled.		
Multiplicity	0..1		
Type	Reference to CanMainFunctionRWPeriods		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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: local

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanHwFilter	0..*	This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter.
CanTTHardwareObjectTrigger	0..*	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.

[SWS_Can_CONSTR_00512] ⌈If the optional parameter CanObjectPayloadLength is configured, the length shall be set that every PDU received or sent via that HOH "fits" into it. Therefore, if set, CanObjectPayloadLength shall be equal or greater than the maximum PduLength of all affected Pdus of the EcuCPduCollection.⌋*()*

Note: For A_HOH that has CanObjectPayloadLength configured and any PDU it sends/receives, A_PDU_Of_A_HOH the condition Can/CanConfigSet/A_HOH/CanObjectPayloadLength >ĒcuC/EcuCPduCollection/A_PDU_Of_A_HOH/PduLength must hold.

10.2.9 CanHwFilter

SWS Item	**[ECUC_Can_00468]**
Container Name	CanHwFilter
Parent Container	CanHardwareObject
Description	This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter.
Configuration Parameters	

SWS Item	**[ECUC_Can_00469]**		
Parameter Name	CanHwFilterCode		
Parent Container	CanHwFilter		
Description	Specifies (together with the filter mask) the identifiers range that passes the hardware filter.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
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			

SWS Item	**[ECUC_Can_00470]**		
Parameter Name	CanHwFilterMask		
Parent Container	CanHwFilter		
Description	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		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
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	dependency: The filter mask settings must be known by the CanIf configuration for optimization of the SW filters.		

No Included Containers

10.2.10 CanConfigSet

SWS Item	**[ECUC_Can_00343]**
Container Name	CanConfigSet
Parent Container	Can
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Can module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanController	1..*	This container contains the configuration parameters of the CAN controller(s).
CanHardwareObject	0..*	This container contains the configuration (parameters) of CAN Hardware Objects.
CanXLHardwareObject	0..*	This container is specified in the SWS CAN XL Driver and contains the configuration (parameters) of CAN XL Hardware Objects.

10.2.11 CanMainFunctionRWPeriods

SWS Item	**[ECUC_Can_00437]**		
Container Name	CanMainFunctionRWPeriods		
Parent Container	CanGeneral		
Description	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.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_Can_00484]**		
Parameter Name	CanMainFunctionPeriod		
Parent Container	CanMainFunctionRWPeriods		
Description	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.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
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		

No Included Containers

10.2.12 CanTTController

SWS Item	**[ECUC_Can_00001]**
Container Name	CanTTController
Parent Container	CanController
Description	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.
Configuration Parameters	

SWS Item	**[ECUC_Can_00139]**		
Parameter Name	CanTTControllerApplWatchdogLimit		
Parent Container	CanTTController		
Description	Defines the maximum time period (unit is 256 times NTU) after which the application has to serve the watchdog.		
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: ECU		

SWS Item	**[ECUC_Can_00138]**		
Parameter Name	CanTTControllerCycleCountMax		
Parent Container	CanTTController		
Description	Defines the value for cycle_count_max. Allowed values: 0x00: 1 basic cycle 0x01: 2 basic cycles 0x03: 4 basic cycles 0x07: 8 basic cycles 0x0F: 16 basic cycles 0x1F: 32 basic cycles 0x3F: 64 basic cycles		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 63		
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: ECU		

SWS Item	**[ECUC_Can_00136]**		
Parameter Name	CanTTControllerExpectedTxTrigger		
Parent Container	CanTTController		
Description	Number of expected_tx_trigger.		
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		

SWS Item	[ECUC_Can_00135]		
Parameter Name	CanTTControllerExternalClockSynchronisation		
Parent Container	CanTTController		
Description	Enables/disables the external clock synchronization. TRUE: External clock synchronization enabled. FALSE: External clock synchronization disabled. This parameter shall only be configurable if parameter CanTTControllerLevel2 equals TRUE.		
Multiplicity	1		
Type	EcucBooleanParamDef		
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: ECU dependency: CanTTControllerLevel2 (ECUC_Can_00131)		

SWS Item	[ECUC_Can_00134]		
Parameter Name	CanTTControllerGlobalTimeFiltering		
Parent Container	CanTTController		
Description	Enables/disables the global time filtering. TRUE: Global time filtering enabled. FALSE: Global time filtering disabled. This parameter shall only be configurable if parameter CanTTControllerLevel2 equals TRUE.		
Multiplicity	1		
Type	EcucBooleanParamDef		
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 dependency: CanTTControllerLevel2 (ECUC_Can_00131)		

SWS Item	[ECUC_Can_00128]		
Parameter Name	CanTTControllerInitialRefOffset		
Parent Container	CanTTController		
Description	Defines the initial value for ref trigger offset.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 127		
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: ECU		

SWS Item	**[ECUC_Can_00140]**		
Parameter Name	CanTTControllerInterruptEnable		
Parent Container	CanTTController		
Description	Enables/disables the respective interrupts. Bit Position set to 1: Enable respective interrupt. Bit Position set to 0: Disable respective interrupt. Bit Position / Interrupt Source: 10: Application Watchdog. 9: Watch Trigger reached. 8: Initialization Watch Trigger reached. 7: Change of Error Level. 6: Tx Overflow. 5: Tx Underflow. 4: Global Time Error. 3: Gap. 2: Start of Cycle. 1: Time Discontinuity. 0: Master State Change. Bit position "1: Time Discontinuity" and "4: Global Time Error" shall only be configurable if parameter CanTTControllerLevel2 equals TRUE.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 1023		
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 dependency: CanTTControllerLevel2 (ECUC_Can_00131)		

SWS Item	**[ECUC_Can_00131]**		
Parameter Name	CanTTControllerLevel2		
Parent Container	CanTTController		
Description	Defines whether Level 2 or Level 1 is used. TRUE: Level 2. FALSE: Level 1. If this parameter is set to FALSE then all parameters with dependency to Can TTControllerLevel2 need not be configured.		
Multiplicity	1		
Type	EcucBooleanParamDef		
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_00141]**	
Parameter Name	CanTTControllerNTUConfig	
Parent Container	CanTTController	
Description	Defines the config value for NTU (network time unit). Value given in microseconds. The value configured shall be greater than 0. Together with the local oscillator period, the TUR (time unit ratio) can be derived from the NTU. This parameter shall only be configurable if parameter CanTTControllerLevel2 equals TRUE.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[0 .. 100]	
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: ECU dependency: CanTTControllerLevel2 (ECUC_Can_00131)		

SWS Item	**[ECUC_Can_00127]**		
Parameter Name	CanTTControllerOperationMode		
Parent Container	CanTTController		
Description	Defines the operation mode.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CAN_TT_EVENT_SYNC_TIME_TRIGGERED	Event-synchronized time triggered operation	
	CAN_TT_EVENT_TRIGGERED	Event triggered operation (normal can operation without time schedule)	
	CAN_TT_TIME_TRIGGERED	Time triggered operation	
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_00132]**		
Parameter Name	CanTTControllerSyncDeviation		
Parent Container	CanTTController		
Description	Defines the maximum synchronization deviation: Given as a percentage value of the NTU (network time unit). The value configured shall be greater than 0. This parameter shall only be configurable if parameter CanTTControllerLevel2 equals TRUE.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 100]		
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 dependency: CanTTControllerLevel2 (ECUC_Can_00131) Synchronisation Deviation <= 2ˆ(CanTTSyncDeviation + 5)		

SWS Item	**[ECUC_Can_00129]**
Parameter Name	CanTTControllerTimeMaster
Parent Container	CanTTController
Description	Defines whether the controller acts as a potential time master. TRUE: Potential time master. FALSE: Time slave.
Multiplicity	1
Type	EcucBooleanParamDef

▽

△

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: ECU		

SWS Item	**[ECUC_Can_00130]**		
Parameter Name	CanTTControllerTimeMasterPriority		
Parent Container	CanTTController		
Description	Defines the time master priority.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 7		
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: ECU		

SWS Item	**[ECUC_Can_00133]**		
Parameter Name	CanTTControllerTURRestore		
Parent Container	CanTTController		
Description	Enables/disables the TUR restore. Note that the value configured for TUR can be derived from the value configured for NTU and the local oscillator preriod. TRUE: TUR restore enabled. FALSE: TUR restore disabled. This parameter shall only be configurable if parameter CanTTControllerLevel2 equals TRUE.		
Multiplicity	1		
Type	EcucBooleanParamDef		
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 dependency: CanTTControllerLevel2 (ECUC_Can_00131)		

SWS Item	**[ECUC_Can_00137]**	
Parameter Name	CanTTControllerTxEnableWindowLength	
Parent Container	CanTTController	
Description	Length of the tx enable window given in CAN bit times. Definition parameter "Can TTControllerTxEnableWindowlength" is used such that: Length of enable window = CanTTControllerTxEnableWindowLength + 1	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 16	

▽

△

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: ECU		

SWS Item	**[ECUC_Can_00158]**		
Parameter Name	CanTTControllerWatchTriggerGapTimeMark		
Parent Container	CanTTController		
Description	watch trigger time mark after a gap		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
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_00157]**		
Parameter Name	CanTTControllerWatchTriggerTimeMark		
Parent Container	CanTTController		
Description	watch trigger time mark		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
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_00142]**		
Parameter Name	CanTTIRQProcessing		
Parent Container	CanTTController		
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.	
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_00493]**		
Parameter Name	CanTTControllerEcucPartitionRef		
Parent Container	CanTTController		
Description	Maps the Time triggered CAN controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the CAN driver is mapped to.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	true		
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		

No Included Containers

10.2.13 CanTTHardwareObjectTrigger

SWS Item	**[ECUC_Can_00002]**
Container Name	CanTTHardwareObjectTrigger
Parent Container	CanHardwareObject
Description	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.
Configuration Parameters	

SWS Item	**[ECUC_Can_00147]**		
Parameter Name	CanTTHardwareObjectBaseCycle		
Parent Container	CanTTHardwareObjectTrigger		
Description	Defines the cycle_offset. CanTTHardwareObjectBaseCycle must be not greater than cycle_count_max.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 63		
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: ECU		

SWS Item	**[ECUC_Can_00148]**		
Parameter Name	CanTTHardwareObjectCycleRepetition		
Parent Container	CanTTHardwareObjectTrigger		
Description	Defines the repeat_factor. CanTTHardwareObjectCycleRepetition shall be a power of two (2), greater than cycle_ offset but not greater than cycle_count_max + 1.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 64		
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: ECU		

SWS Item	**[ECUC_Can_00146]**		
Parameter Name	CanTTHardwareObjectTimeMark		
Parent Container	CanTTHardwareObjectTrigger		
Description	Defines the point in time, when the trigger will be activated. Value is given in cycle time.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
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_00155]**		
Parameter Name	CanTTHardwareObjectTriggerId		
Parent Container	CanTTHardwareObjectTrigger		
Description	Sequential number which allows separation of different TTCAN triggers configured for one and the same hardware object.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 63		
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 withAuto = true

SWS Item	**[ECUC_Can_00145]**		
Parameter Name	CanTTHardwareObjectTriggerType		
Parent Container	CanTTHardwareObjectTrigger		
Description	Defines the type of the trigger associated with the hardware object. This parameter depends on plain CAN parameter CAN_OBJECT_TYPE. If CAN_OBJECT_TYPE equals RECEIVE than this parameter is fixed to CAN_TT_RX_TRIGGER. If CAN_OBJECT_TYPE equals TRANSMIT than one of the following literals is configurable: CAN_TT_TX_REF_TRIGGER, CAN_TT_TX_REF_TRIGGER_GAP, CAN_TT_TX_TRIGGER_MERGED, CAN_TT_TX_TRIGGER_SINGLE, CAN_TT_TX_TRIGGER_EXCLUSIVE.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CAN_TT_RX_TRIGGER	Trigger for verifying the successful reception of messages.	
	CAN_TT_TX_REF_TRIGGER	Trigger for transmitting the reference message.	
	CAN_TT_TX_REF_TRIGGER_GAP	Trigger for transmitting the reference message in case no event occurs after a gap.	
	CAN_TT_TX_TRIGGER_EXCLUSIVE	Trigger for transmitting a message in an exclusive time window. Note, that messages in an exclusive window are transmitted continuously, i.e. regardless whether the same message has been transmitted before, the message, which is currently available, will be transmitted every time the tx trigger occurs.	
	CAN_TT_TX_TRIGGER_MERGED	Trigger for transmitting a message inside a merged arbitration window (the last tx trigger in a merged arbitration window is of type CAN_TT_TX_TRIGGER_SINGLE). Note, that messages in an arbitration window are transmitted only, if new data is available. When the transmission was not successful, it will be repeated at the next tx trigger for this message. When the transmission was successful, this message will not be transmitted again at the next tx triggers until a new message for this tx trigger is provided.	
	CAN_TT_TX_TRIGGER_SINGLE	Trigger for transmitting a message in a single (non-merged) arbitration window (or the last tx trigger in a merged arbitration window). Note, that messages in an arbitration window are transmitted only, if new data is available. When the transmission was not successful, it will be repeated at the next tx trigger for this message. When the transmission was successful, this message will not be transmitted again at the next tx triggers until a new message for this tx trigger is provided.	
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 dependency: CAN_OBJECT_TYPE		

No Included Containers

10.2.14 CanXLGeneral

SWS Item	**[ECUC_Can_00524]**		
Container Name	CanXLGeneral		
Parent Container	CanGeneral		
Description	This container is specified in the SWS CAN XL Driver and contains global parameters of the CAN XL Driver.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_Can_00525]**		
Parameter Name	CanXLEthGlobalTimeSupport		
Parent Container	CanXLGeneral		
Description	Enables/Disables the Global Time APIs for the Ethernet Interface used when hardware timestamping is supported by CAN controller.		
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		

No Included Containers

10.2.15 CanXLController

SWS Item	**[ECUC_Can_00499]**
Container Name	CanXLController
Parent Container	CanController
Description	This container is specified in the SWS CAN XL Driver and represents a CAN XL channel. If this container is present, the CAN driver will provide the extended CanXL API.
Post-Build Variant Multiplicity	false

▽

△

Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

SWS Item	**[ECUC_Can_00500]**		
Parameter Name	CanXLCtrlEthDefaultPriority		
Parent Container	CanXLController		
Description	Defines the default CAN XL Priority ID to be used for outgoing tunneled Ethernet frames.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
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_00501]**		
Parameter Name	CanXLEthDefaultQueue		
Parent Container	CanXLController		
Description	Defines the default CAN XL Queue to be used for outgoing tunneled Ethernet frames.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
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_00506]**
Parameter Name	CanXLEthPhysAddress
Parent Container	CanXLController
Description	Specifies the unique 48-bit physical address (MAC address) of the controller in network byte order.

▽

△

Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	–		
Length	17-17		
Regular Expression	([0-9a-fA-F]\{2}:)\{5}[0-9a-fA-F]\{2}		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
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_00511]**		
Parameter Name	CanXLEthEcucPartitionRef		
Parent Container	CanXLController		
Description	Maps the Ethernet Interface access to the CAN XL controller to zero or one ECUC partitions.		
Multiplicity	0..1		
Type	Reference to EcucPartition		
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: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanXLEthEgressFifo	0..*	Represents a Fifo at the egress side.
CanXLEthIngressFifo	0..*	Represents a Fifo at the ingress side.

10.2.16 CanXLHardwareObject

SWS Item	**[ECUC_Can_00526]**
Container Name	CanXLHardwareObject
Parent Container	CanConfigSet
Description	This container is specified in the SWS CAN XL Driver and contains the configuration (parameters) of CAN XL Hardware Objects.

▽

△

Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**[ECUC_Can_00327]**		
Parameter Name	CanObjectType		
Parent Container	CanXLHardwareObject		
Description	Specifies if the HardwareObject is used as Transmit or as Receive object		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RECEIVE	Receive HOH	
	TRANSMIT	Transmit HOH	
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_00527]**		
Parameter Name	CanXLObjectId		
Parent Container	CanXLHardwareObject		
Description	Holds the handle ID of CAN XL HRH or HTH.		
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: local		

SWS Item	**[ECUC_Can_00322]**		
Parameter Name	CanControllerRef		
Parent Container	CanXLHardwareObject		
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**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: The referenced CanController has to contain a CanXLController.		

SWS Item	**[ECUC_Can_00438]**		
Parameter Name	CanMainFunctionRWPeriodRef		
Parent Container	CanXLHardwareObject		
Description	Reference to CanMainFunctionPeriod. If configured, this hardware object will be polled.		
Multiplicity	0..1		
Type	Reference to CanMainFunctionRWPeriods		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanXLHwFilter	0..*	This container is only valid for CAN XL HRHs and contains the configuration (parameters) of one hardware filter. This container is intentionally left empty, because the parameters are very hardware specific and shall be filled in by the VSMD.

10.2.17 CanXLHwFilter

SWS Item	**[ECUC_Can_00528]**		
Container Name	CanXLHwFilter		
Parent Container	CanXLHardwareObject		
Description	This container is only valid for CAN XL HRHs and contains the configuration (parameters) of one hardware filter. This container is intentionally left empty, because the parameters are very hardware specific and shall be filled in by the VSMD.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

No Included Containers

10.2.18 CanXLBaudrateConfig

SWS Item	**[ECUC_Can_00512]**		
Container Name	CanXLBaudrateConfig		
Parent Container	CanControllerBaudrateConfig		
Description	This container is specified in the SWS CAN XL Driver and contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN XL frame.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**[ECUC_Can_00513]**		
Parameter Name	CanXLBaudRate		
Parent Container	CanXLBaudrateConfig		
Description	Specifies the data segment baud rate of the controller in kbps. Note: The CAN XL baudrate should be at least twice the nominal bitrate so that an error flag can safely destroy a CAN XL frame.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 20000]		
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 dependency: Has to be at least twice as high as CanControllerBaudRate.		

SWS Item	**[ECUC_Can_00523]**		
Parameter Name	CanXLErrorSignaling		
Parent Container	CanXLBaudrateConfig		
Description	Specifies if error signaling shall be enabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
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 dependency: Only relevant if CanXLTrcvPwmMode is disabled.		

SWS Item	**[ECUC_Can_00517]**
Parameter Name	CanXLPropSeg
Parent Container	CanXLBaudrateConfig

▽

△

Description	Specifies propagation delay 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		

SWS Item	**[ECUC_Can_00514]**		
Parameter Name	CanXLPwmL		
Parent Container	CanXLBaudrateConfig		
Description	Specifies the PWM long phase length.		
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		

SWS Item	**[ECUC_Can_00516]**		
Parameter Name	CanXLPwmO		
Parent Container	CanXLBaudrateConfig		
Description	Specifies the PWM time offset.		
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		

SWS Item	**[ECUC_Can_00515]**	
Parameter Name	CanXLPwmS	
Parent Container	CanXLBaudrateConfig	
Description	Specifies the PWM short phase length.	
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		

SWS Item	**[ECUC_Can_00518]**		
Parameter Name	CanXLSeg1		
Parent Container	CanXLBaudrateConfig		
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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00519]**		
Parameter Name	CanXLSeg2		
Parent Container	CanXLBaudrateConfig		
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	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**[ECUC_Can_00521]**	
Parameter Name	CanXLSspOffset	
Parent Container	CanXLBaudrateConfig	
Description	Specifies the Transmitter Delay Compensation Offset in minimum time quanta. If this parameter is configured, the Transmitter Delay Compensation is done by measurement of the CAN controller. If not specified, Transmitter Delay Compensation is disabled. See ECUC_Can_00494 for details.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 255	
Default value	–	
Post-Build Variant Multiplicity	true	

▽

△

Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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: local		

SWS Item	**[ECUC_Can_00520]**		
Parameter Name	CanXLSyncJumpWidth		
Parent Container	CanXLBaudrateConfig		
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		

SWS Item	**[ECUC_Can_00522]**		
Parameter Name	CanXLTrcvPwmMode		
Parent Container	CanXLBaudrateConfig		
Description	Specifies if the transceiver shall be set to the PWM mode.		
Multiplicity	1		
Type	EcucBooleanParamDef		
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		

No Included Containers

10.2.19 CanXLEthEgressFifo

SWS Item	**[ECUC_Can_00502]**
Container Name	CanXLEthEgressFifo
Parent Container	CanXLController
Description	Represents a Fifo at the egress side.

▽

△

Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**[ECUC_Can_00503]**		
Parameter Name	CanXLEthEgressFifoCanXLPriority		
Parent Container	CanXLEthEgressFifo		
Description	Defines the CAN XL Priority ID to be used for outgoing tunneled Ethernet frames using this FIFO.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
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_00504]**		
Parameter Name	CanXLEthEgressFifoCanXLQueue		
Parent Container	CanXLEthEgressFifo		
Description	Defines the CAN XL Queue to be used for outgoing tunneled Ethernet frames using this FIFO.		
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		

SWS Item	**[ECUC_Can_00505]**		
Parameter Name	CanXLEthEgressFifoIdx		
Parent Container	CanXLEthEgressFifo		
Description	Egress Fifo index.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
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 withAuto = true		

No Included Containers

10.2.20 CanXLEthIngressFifo

SWS Item	**[ECUC_Can_00507]**		
Container Name	CanXLEthIngressFifo		
Parent Container	CanXLController		
Description	Represents a Fifo at the ingress side.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

SWS Item	**[ECUC_Can_00509]**		
Parameter Name	CanXLEthIngressFifoCanXLQueue		
Parent Container	CanXLEthIngressFifo		
Description	Defines the CAN XL Queue to be used for incoming tunneled Ethernet frames using this FIFO.		
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		

SWS Item	**[ECUC_Can_00508]**		
Parameter Name	CanXLEthIngressFifoIdx		
Parent Container	CanXLEthIngressFifo		
Description	Ingress Fifo index.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
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 withAuto = true		

SWS Item	**[ECUC_Can_00510]**		
Parameter Name	CanXLEthIngressFifoVcid		
Parent Container	CanXLEthIngressFifo		
Description	Configures a VCID to be accepted by this FIFO. If not present, all VCIDs shall be accepted.		
Multiplicity	0..*		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	–		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	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: local		

No Included Containers

11 Not applicable requirements

[SWS_Can_NA] ⌈These requirements are not applicable to this specification.⌋*(SRS_-BSW_00162, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00307, SRS_-BSW_00325, SRS_BSW_00336, SRS_BSW_00342, SRS_BSW_00353, SRS_-BSW_00359, SRS_BSW_00378, SRS_BSW_00383, SRS_BSW_00395, SRS_-BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_-BSW_00409, SRS_BSW_00413, SRS_BSW_00415, SRS_BSW_00417, SRS_-BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_-BSW_00426, SRS_BSW_00427, SRS_BSW_00429, SRS_BSW_00433, SRS_-BSW_00439, SRS_BSW_00440, SRS_BSW_00447, SRS_BSW_00449, SRS_-BSW_00453, SRS_Can_01125, SRS_Can_01126, SRS_SPAL_12064, SRS_SPAL_-12068, SRS_SPAL_12163, SRS_SPAL_12462)*