

---

# User Manual

for S32K14X CAN Driver

Document Number: UM2CANASR4.3 Rev0001R1.0.1  
Rev. 1.0





# Contents

Section number	Title	Page
<b>Chapter 1</b>		
<b>Revision History</b>		
<b>Chapter 2</b>		
<b>Introduction</b>		
2.1	Supported Derivatives.....	15
2.2	Overview.....	15
2.3	About this Manual.....	16
2.4	Acronyms and Definitions.....	16
2.5	Reference List.....	17
<b>Chapter 3</b>		
<b>Driver</b>		
3.1	Requirements.....	19
3.2	Driver Design Sumary.....	19
3.3	Hardware Resources.....	20
3.4	Deviation from Requirements.....	20
3.5	Driver Limitation.....	24
3.6	Driver Usage and Configuration Tips.....	25
3.6.1	Driver APIs usage.....	25
3.6.2	Can bit timing configuration.....	25
3.6.2.1	Can bit time calculation.....	26
3.6.2.2	Can FD bit time configuration.....	28
3.6.2.3	Can CBT usage and configuration.....	28
3.6.3	Can Hardware Object Handle (HOH) configuration.....	28
3.6.3.1	Configuration for Can FD feature.....	29
3.6.3.2	HOH configuration for Rx FIFO feature.....	29
3.6.4	Rx Fifo configuration.....	30
3.7	Runtime Errors.....	32
3.8	Software specification.....	32

Section number	Title	Page
3.8.1	Define Reference.....	33
3.8.1.1	Define CAN_API_ENABLE_ABORT_MB.....	33
3.8.1.2	Define CAN_BCC_SUPPORT_ENABLE.....	33
3.8.1.3	Define CAN_BUSOFFPOLLSUPPORTED.....	33
3.8.1.4	Define CAN_SET_BAUDRATE_API.....	33
3.8.1.5	Define CAN_DEV_ERROR_DETECT.....	33
3.8.1.6	Define CAN_DUAL_CLOCK_MODE.....	34
3.8.1.7	Define CAN_E_DEFAULT.....	34
3.8.1.8	Define CAN_E_DATALOST.....	34
3.8.1.9	Define CAN_E_PARAM_CONTROLLER.....	34
3.8.1.10	Define CAN_E_PARAM_DATA_LENGTH.....	35
3.8.1.11	Define CAN_E_PARAM_HANDLE.....	35
3.8.1.12	Define CAN_E_PARAM_POINTER.....	35
3.8.1.13	Define CAN_E_TRANSITION.....	36
3.8.1.14	Define CAN_E_UNINIT.....	36
3.8.1.15	Define CAN_E_PARAM_BAUDRATE.....	37
3.8.1.16	Define CAN_ERROR_NOTIFICATION_ENABLE.....	37
3.8.1.17	Define CAN_EXTENDEDID.....	37
3.8.1.18	Define CAN_INIT_CONFIG_CONTROLLERS_PC.....	37
3.8.1.19	Define CAN_INIT_CONFIG_PC_DEFINES.....	37
3.8.1.20	Define CAN_ISROPTCODESIZE.....	38
3.8.1.21	Define CAN_MAINFUNCTION_MODE_PERIOD.....	38
3.8.1.22	Define CAN_MAINFUNCTION_PERIOD_BUSOFF.....	38
3.8.1.23	Define CAN_MAINFUNCTION_PERIOD_READ.....	38
3.8.1.24	Define CAN_MAINFUNCTION_PERIOD_WRITE.....	38
3.8.1.25	Define CAN_MAXCTRL_SUPPORTED.....	38
3.8.1.26	Define CAN_MAXMB_SUPPORTED.....	39
3.8.1.27	Define CAN_MBCOUNTTEXTENSION.....	39
3.8.1.28	Define CAN_MIX_MB_SUPPORT.....	39

Section number	Title	Page
3.8.1.29	Define CAN_MULTIPLEXED_TRANSMISSION.....	39
3.8.1.30	Define CAN_PRECOMPILE_SUPPORT.....	39
3.8.1.31	Define CAN_RXFIFO_ENABLE.....	39
3.8.1.32	Define CAN_RXFIFO_EVENT_UNIFIED.....	39
3.8.1.33	Define CAN_RXPOLL_SUPPORTED.....	40
3.8.1.34	Define CAN_SID_ABORT_MB.....	40
3.8.1.35	Define CAN_SID_DISABLE_CONTROLLER_INTERRUPTS.....	40
3.8.1.36	Define CAN_SID_ENABLE_CONTROLLER_INTERRUPTS.....	40
3.8.1.37	Define CAN_SID_GET_VERSION_INFO.....	40
3.8.1.38	Define CAN_SID_INIT.....	40
3.8.1.39	Define CAN_SID_MAIN_FUNCTION_BUS_OFF.....	41
3.8.1.40	Define CAN_SID_MAIN_FUNCTION_MODE.....	41
3.8.1.41	Define CAN_SID_MAIN_FUNCTION_READ.....	41
3.8.1.42	Define CAN_SID_MAIN_FUNCTION_WAKEUP.....	41
3.8.1.43	Define CAN_SID_MAIN_FUNCTION_WRITE.....	41
3.8.1.44	Define CAN_SID_SET_CONTROLLER_MODE.....	41
3.8.1.45	Define CAN_SID_SETCLOCKMODE.....	42
3.8.1.46	Define CAN_SID_WRITE.....	42
3.8.1.47	Define CAN_SID_SET_BAUDRATE.....	42
3.8.1.48	Define CAN_SET_BAUDRATE_API.....	33
3.8.1.49	Define CAN_TIMEOUT_DURATION.....	42
3.8.1.50	Define CAN_TXPOLL_SUPPORTED.....	42
3.8.1.51	Define CAN_VERSION_INFO_API.....	43
3.8.2	Enum Reference.....	43
3.8.2.1	Enumeration Can_ClockModeType.....	43
3.8.2.2	Enumeration Can_ControllerStateType.....	43
3.8.2.3	Enumeration Can_ObjType.....	43
3.8.2.4	Enumeration Can_ReturnType.....	44
3.8.2.5	Enumeration Can_ControllerStateType.....	44

Section number	Title	Page
3.8.2.6	Enumeration Can_StatusType.....	44
3.8.2.7	Enumeration CanIdType.....	45
3.8.2.8	Enumeration Can_FdType.....	45
3.8.3	Function Reference.....	45
3.8.3.1	Function Can_Init.....	46
3.8.3.2	Function Can_GetVersionInfo.....	46
3.8.3.3	Function Can_SetControllerMode.....	47
3.8.3.4	Function Can_DisableControllerInterrupts.....	47
3.8.3.5	Function Can_EnableControllerInterrupts.....	48
3.8.3.6	Function Can_Write.....	48
3.8.3.7	Function Can_MainFunction_Write.....	49
3.8.3.8	Function Can_MainFunction_Read.....	50
3.8.3.9	Function Can_MainFunction_BusOff.....	50
3.8.3.10	Function Can_MainFunction_Mode.....	51
3.8.3.11	Function Can_AbortMb.....	51
3.8.3.12	Function Can_SetClockMode.....	51
3.8.3.13	Function Can_SetBaudrate.....	52
3.8.3.14	Function Can_DeInit.....	53
3.8.3.15	Function Can_GetControllerMode.....	53
3.8.3.16	Function Can_GetControllerErrorState.....	54
3.8.4	Structs Reference.....	54
3.8.4.1	Structure Can_ConfigType and CanStatic_ConfigType.....	54
3.8.4.2	Structure Can_ControllerDescriptorType.....	55
3.8.4.3	Structure Can_ControllerBaudrateConfigType and CanStatic_ControllerBaudrateConfigType.....	56
3.8.4.4	Structure Can_ControllerBaudrateConfigType.....	57
3.8.4.5	Structure Can_MBCConfigContainerType.....	58
3.8.4.6	Structure Can_MBCConfigObjectType.....	59
3.8.4.7	Structure Can_PduType.....	60
3.8.4.8	Structure Can_RxFifoTableIdConfigType.....	60

Section number	Title	Page
3.8.4.9	Structure Can_ControllerFdConfigType.....	61
3.8.4.10	Structure Can_ControllerCbtConfigType.....	61
3.8.5	Types Reference.....	62
3.8.5.1	Typedef Can_HwHandleType.....	62
3.8.5.2	Typedef Can_IdPtrType.....	62
3.8.5.3	Typedef Can_IdType.....	63
3.8.5.4	Typedef Can_PCallBackType.....	63
3.8.5.5	Typedef Can_PtrControlerDescriptorType.....	63
3.8.5.6	Typedef CanStatic_PtrControlerDescriptorType.....	63
3.8.5.7	Typedef Can_PtrMBConfigContainerType.....	63
3.9	Symbolic Names Disclaimer.....	63

## Chapter 4

### Tresos Configuration Plug-in

4.1	Configuration elements of Can.....	65
4.2	Form IMPLEMENTATION_CONFIG_VARIANT.....	65
4.3	Form CanController.....	66
4.3.1	Can-Controller Parameters.....	66
4.3.1.1	CanHwChannel.....	67
4.3.1.2	CanControllerActivation.....	68
4.3.1.3	CanControllerBaseAddress.....	69
4.3.1.4	CanControllerId.....	69
4.3.1.5	CanRxProcessing.....	69
4.3.1.6	CanTxProcessing.....	70
4.3.1.7	CanBusOffProcessing.....	71
4.3.1.8	CanListenOnlyMode.....	72
4.3.1.9	CanLoopBackMode.....	72
4.3.1.10	CanSoftwareBusOffRecovery.....	73
4.3.1.11	CanAutoBusOffRecovery.....	74
4.3.1.12	CanTripleSamplingEnable.....	75

Section number	Title	Page
4.3.1.13	CanLowestBuffTransmitFirst.....	75
4.3.1.14	CanLocalPriorityEn.....	76
4.3.1.15	CanClockFromBus.....	77
4.3.1.16	CanCpuClockRef.....	77
4.3.1.17	CanControllerRXFifoEnable.....	78
4.3.1.18	CanRxFifoWarningNotification.....	78
4.3.1.19	CanRxFifoOverflowNotification.....	79
4.3.1.20	CanErrorControllerNotification.....	79
4.3.1.21	CanCpuClockRef_Alternate.....	79
4.3.1.22	CanBccSupport.....	80
4.3.1.23	CanErrorControllerNotifEn.....	81
4.3.1.24	CanControllerDefaultBaudrate.....	82
4.3.1.25	CanControllerBaudrateConfig.....	82
4.3.1.25.1	CanControllerCheckCanStandard.....	83
4.3.1.25.2	CanAdvancedSetting.....	84
4.3.1.25.3	CanBusLength.....	84
4.3.1.25.4	CanPropDelayTranceiver.....	85
4.3.1.25.5	CanTxArbitrationStartDelay.....	85
4.3.1.25.6	CanControllerPrescaller.....	85
4.3.1.25.7	CanControllerPrescaller_Alt.....	86
4.3.1.25.8	CanControllerBaudRate.....	86
4.3.1.25.9	CanControllerPropSeg.....	87
4.3.1.25.10	CanControllerSeg1.....	87
4.3.1.25.11	CanControllerSeg2.....	88
4.3.1.25.12	CanControllerSyncJumpWidth.....	88
4.3.1.25.13	CanControllerFdBaudrateConfig.....	89
4.3.1.25.14	CanControllerFdBaudRate.....	89
4.3.1.25.15	CanControllerPropSeg.....	90
4.3.1.25.16	CanControllerFDPrescaller.....	90



Section number	Title	Page
4.3.1.25.17	CanControllerSeg1.....	91
4.3.1.25.18	CanControllerSeg2.....	91
4.3.1.25.19	CanControllerSyncJumpWidth.....	92
4.3.1.25.20	CanControllerTrcvDelayCompensationOffset.....	93
4.3.1.25.21	CanControllerTxBitRateSwitch.....	93
4.3.1.25.22	CanControllerCbtEnable.....	94
4.3.1.25.23	CanControllerBaudRate.....	94
4.3.1.25.24	CanControllerCbtPropSeg.....	95
4.3.1.25.25	CanControllerCbtPrescaller.....	95
4.3.1.25.26	CanControllerCbtSeg1.....	96
4.3.1.25.27	CanControllerCbtSeg2.....	97
4.3.1.25.28	CanControllerSyncJumpWidthCbt.....	97
4.3.1.25.29	CanControllerFdIsoCANFD.....	98
4.3.1.25.30	CanControllerFdPrExcEn.....	98
4.3.1.25.31	CanControllerFdEdgeFilterDis.....	98
4.3.1.26	Can RxFifo.....	99
4.3.1.26.1	CanControllerIDAcceptanceMode.....	102
4.3.1.26.2	CanIDValue0.....	102
4.3.1.26.3	CanIDValue1.....	103
4.3.1.26.4	CanIDValue2.....	104
4.3.1.26.5	CanIDValue3.....	105
4.3.1.26.6	CanMBFilterMaskValue.....	105
4.3.1.26.7	CanTableIDType.....	106
4.3.2	Can HardwareObject.....	106
4.3.2.1	CanFdPaddingValue.....	107
4.3.2.2	CanHandleType.....	108
4.3.2.3	CanIdType.....	108
4.3.2.4	CanIdValue.....	109
4.3.2.5	CanMBPrio.....	110

Section number	Title	Page
4.3.2.6	CanObjectId.....	111
4.3.2.7	CanObjectType.....	111
4.3.2.8	CanControllerRef.....	112
4.3.2.9	CanFilterMaskRef.....	113
4.3.2.10	CanMainFunctionRWPeriodRef.....	114
4.3.3	CanFilterMask.....	114
4.3.3.1	CanHwFilterMask.....	115
4.3.3.2	CanHwFilterCode.....	115
4.3.4	Can-General Parameters.....	116
4.3.4.1	CanDevErrorDetect.....	116
4.3.4.2	CanVersionInfoApi.....	117
4.3.4.3	CanIndex.....	117
4.3.4.4	CanMainFunctionBusOffPeriod.....	117
4.3.4.5	CanMainFunctionModePeriod.....	118
4.3.4.6	CanMultiplexedTransmission.....	118
4.3.4.7	CanTimeoutDuration.....	119
4.3.4.8	CanLPduReceiveCalloutFunction.....	119
4.3.4.9	CanCodeSizeOptimization.....	120
4.3.4.10	CanExtendedIdSupport.....	120
4.3.4.11	CanMBCountExtensionSupport.....	121
4.3.4.12	CanApiEnableMbAbort.....	121
4.3.4.13	CanEnableDualClockMode.....	122
4.3.4.14	CanCounterRef.....	122
4.3.4.15	CanSupportTTCANRef.....	123
4.3.4.16	CanMainFunctionRWPeriods.....	123
4.3.4.16.1	CanMainFunctionPeriod.....	124
4.4	Form CommonPublishedInformation.....	124
4.4.1	ArReleaseMajorVersion (CommonPublishedInformation).....	125
4.4.2	ArReleaseMinorVersion (CommonPublishedInformation).....	125

Section number	Title	Page
4.4.3	ArReleaseRevisionVersion (CommonPublishedInformation).....	126
4.4.4	ModuleId (CommonPublishedInformation).....	126
4.4.5	SwMajorVersion (CommonPublishedInformation).....	127
4.4.6	SwMinorVersion (CommonPublishedInformation).....	127
4.4.7	SwPatchVersion (CommonPublishedInformation).....	128
4.4.8	VendorApiInfix (CommonPublishedInformation).....	128
4.4.9	VendorId (CommonPublishedInformation).....	128



# Chapter 1

## Revision History

**Table 1-1. Revision History**

Revision	Date	Author	Description
1.0	21/06/2019	NXP MCAL Team	Updated version for ASR 4.3.1S32K14XR1.0.1



# Chapter 2

## Introduction

This User Manual describes NXP Semiconductors AUTOSAR Controller Area Network ( CAN ) for S32K14X .

AUTOSAR CAN driver configuration parameters and deviations from the specification are described in CAN Driver chapter of this document. AUTOSAR CAN driver requirements and APIs are described in the AUTOSAR CAN driver software specification document.

### 2.1 Supported Derivatives

The software described in this document is intended to be used with the following microcontroller devices of NXP Semiconductors .

**Table 2-1. S32K14X Derivatives**

NXP Semiconductors	s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64, s32k142_lqfp48, s32k144_lqfp48, s32k148_lqfp100
--------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

All of the above microcontroller devices are collectively named as S32K14X .

### 2.2 Overview

**AUTOSAR (AUTomotive Open System ARchitecture)** is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

## AUTOSAR

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

## 2.3 About this Manual

This Technical Reference employs the following typographical conventions:

**Boldface type:** Bold is used for important terms, notes and warnings.

*Italic font:* Italic typeface is used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

### Note

This is a note.

## 2.4 Acronyms and Definitions

**Table 2-2. Acronyms and Definitions**

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
ASM	Assembler
BSMI	Basic Software Make file Interface
CAN	Controller Area Network
DEM	Diagnostic Event Manager
DET	Development Error Tracer
C/CPP	C and C++ Source Code
VLE	Variable Length Encoding

*Table continues on the next page...*



**Table 2-2. Acronyms and Definitions (continued)**

Term	Definition
N/A	Not Applicable
MCU	Micro Controller Unit

## 2.5 Reference List

**Table 2-3. Reference List**

#	Title	Version
1	Specification of CAN Driver	AUTOSAR Release 4.3.1
2	S32K14X Reference Manual	Reference Manual, Rev. 9, 9/2018
3	S32K142 Mask Set Errata for Mask 0N33V (0N33V)	30/11/2017
4	S32K144 Mask Set Errata for Mask 0N57U (0N57U)	30/11/2017
5	S32K146 Mask Set Errata for Mask 0N73V (0N73V)	30/11/2017
6	S32K148 Mask Set Errata for Mask 0N20V (0N20V)	25/10/2018
7	S32K118 Mask Set Errata for Mask 0N97V (0N97V)	07/01/2019



## Chapter 3 Driver

### 3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.3 Rev0001CAN Driver Software Specification document (See Table [Reference List](#) ).

### 3.2 Driver Design Summary

The S32K14X contains up to 3 Controller Area Network (CAN) blocks. Which support CAN FD.

Each IPV\_FlexCAN module is a full implementation of the CAN protocol specification, the CAN with Flexible Data rate (CAN FD) protocol and the CAN 2.0 version B protocol. The CAN protocol interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The message buffer management (MBM) sub-module handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The bus interface unit (BIU) sub-module controls the access to and from the internal interface bus, to establish connection to the CPU and other blocks.

FlexCAN\_0 includes a new FlexCAN module feature called Pretended Networking mode that addresses the low power requirements of applications. Pretended Networking mode adds specific wake up functionality in low power mode (Stop mode) including wake up by a successful filtered Rx message or a timeout event.

The IPV\_FlexCAN has these major features:

- 32 flexible message buffers (MBs) of zero to eight bytes data length. With CAN\_FD, this length is from 0 to 64 bytes. Some platform has support the selecting ISO/non-ISO.

- Individual Rx mask registers per message buffer.
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 128 Extended, 256 Standard, or 512 Partial (8 bits) IDs, with 32 individual masking capability.
- ListenOnly capability.
- Programmable loop-back mode supporting self-test operation.
- Maskable interrupts.
- Low power modes.
- Programmable clock source to the CAN Protocol Interface, either peripheral clock or oscillator clock.
- Transceiver Delay Compensation feature when transmitting CAN FD messages at faster data rates.
- Supports Pretended Networking functionality in low power: Stop mode.

### 3.3 Hardware Resources

None.

The CAN controller number mapping between Reference Manual/microcontroller and our XDM configuration can be done by using the following:

- Reference Manual naming = Configuration Naming
- FlexCAN\_0 = FlexCAN\_A
- FlexCAN\_1 = FlexCAN\_B
- FlexCAN\_2 = FlexCAN\_C

### 3.4 Deviation from Requirements

The driver deviates from the AUTOSAR CAN Driver software specification in some places.

There are also some additional requirements (on top of requirements detailed in AUTOSAR CAN Driver software specification) which need to be satisfied for correct operation.

1. The driver does not distinguish between “Extended” and “Mixed” MB types for receiving way: All Rx MBs configured as MIXED type will be converted to EXTENDED type. For transmission the CanIf will prepare the message ID with MSB bit set and based on this fact the Can module will send the message as STANDARD or EXTENDED type. See CANIF188 and CANIF281 requirements.

2. Priority inversion may occur even if Cancellation and Transmission multiplexing is enabled. When all message buffers in the transmission pool are full and scheduled for transmission a new call to Can\_Write will cause a search which identifies the lowest priority message. If the lowest priority message has lower priority than the new message submitted to Can\_Write, then cancellation of the message currently stored in the message buffer will be attempted. There is, however, a possibility that this lowest priority message might be successfully transmitted after the Can\_Write has read the message buffer during its search. If a new high priority message is immediately scheduled for transmission (via preemptive call to Can\_Write) the identification of the message buffer holding the lowest priority message will no longer be correct (in the underlying Can\_Write which has been preempted). This may lead to the message not being cancelled (as it now may have a higher priority). In this case Can\_Write will not repeat the search for the lowest priority message and priority inversion may occur (if there is another message with lower priority scheduled for transmission in a different message buffer). Whether this scenario can or cannot occur in a particular application depends on implementation of the CanIf.
3. The driver does not depend of the OS to get a timer value.
4. For functions that are blocking(need to wait a limited period of time for something to happen) the time duration is not given in secons, but in loops.
5. There is only one present for the polling functions Can\_MainFunctions\_Read and Can\_MainFunctions\_Write, because there is only one reference of the HardwareObject to do polling.
6. The base address for the controllers is not user input.

Table [Table 3-1](#) provides Status column description.

**Table 3-1. Deviations Status Column Description**

Term	Definition
N/A	Not available
N/T	Not testable
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently, or out of scope for the CAN driver.

**Table 3-2. CAN Deviations Table**

Requirement	Status	Description	Notes
SWS_Can_00 447	N/S	Icu_DisableNotification shall be called when "external" Can controllers have been transitioned to STOPPED state (CANIF_CS_STOPPED).	All controllers are on chip.
SWS_Can_00 242	N/S	If an off-chip CAN controller is used, the driver uses services of other MCAL drivers (i.e. SPI). These drivers need to be up and running before the CAN controller can be initialized. The sequence of initialization of different drivers is partly specified in [7]. Only Synchronous APIs may be used because the CAN driver does not provide callback functions that can be called by the MCAL driver. Thus the type of connection between $\mu$ C and CAN Hardware Unit has only impact on implementation and not on the API.	No off-chip controller is used.
SWS_Can_00 110	N/S	There is no requirement regarding the execution order of the CAN main processing functions.	Application Code Requirement.
SWS_Can_00 240	N/S	The Can module's environment shall make sure that the Mcu module is initialized before initializing the Can module.	Can driver cannot access any variable of Mcu module for checking the state.
SWS_Can_00 077	N/F	For CAN Hardware Units of different type, different Can modules shall be implemented.	Current platforms have only one type of hardware unit.
ECUC_Can_0 0382	N/S	Specifies the CAN controller base address.	The possibility for user to input the base address of the controller is not supported by the Can Driver. The base address is given in the Base module.
SWS_Can_00 244	N/S	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.	All current controllers are onchip.No callaback can be called by other drivers.
SWS_Can_00 391	N/S	Can module implementations for off-chip CAN controllers shall include the header file Spi.h. By this inclusion, the APIs to access an external CAN controller by the SPI module [12] are included.	Controllers are onchip.
SWS_Can_00 257	N/S	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.	Controller not support sleep mode.
SWS_Can_00 270	N/S	On hardware wakeup (triggered by a wake-up event from CAN bus), the CAN controller shall transition into the state STOPPED.	Platform does not support a WAKE UP mode.
SWS_Can_00 271	N/S	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.	Platform does not support a WAKE UP mode.
SWS_Can_00 269	N/S	The Can module shall not further process the L-PDU that caused a wake-up.	Platform does not support a WAKE UP mode.
SWS_Can_00 048	N/S	In case of a CAN bus wake-up during sleep transition, the function	For HW not support on-chip wakeup, this event will not occur.

*Table continues on the next page...*

**Table 3-2. CAN Deviations Table (continued)**

Requirement	Status	Description	Notes
		Can_SetControllerMode(CAN_T_WAKEUP) shall return CAN_NOT_OK.	
SWS_Can_00 274	N/S	The Can module shall disable or suppress automatic bus-off recovery.	it is replaced by PR-MCAL-3014: The CAN driver configuration shall allow automatic and also manual CAN bus-off recovery.
SWS_CAN_00 490	N/S	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".?	Hardware support FIFO, so this requirement is not applicable.
SWS_Can_00 364	N/S	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.	Only for platform support wake up.
SWS_Can_00 294	N/S	The function Can_SetControllerMode shall disable the wake-up interrupt, while checking the wake-up status.	Platform does not support a WAKE UP mode.
SWS_Can_00 361	N/S	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.	AAI-279: reject this requirement because our SoC does not support wakeup.
SWS_Can_00 445	N/S	Can driver shall use the following APIs provided by Icu driver, to enable and disable the wakeup event notification: Icu_EnableNotification Icu_DisableNotification.	Not implemented for OnChip platform.
SWS_Can_00 446	N/S	Icu_EnableNotification shall be called when "external" CAN controllers have been transitioned to SLEEP state (CANIF_CS_SLEEP).	Not implemented for OnChip platform.
SWS_Can_00 228	N/S	The Can_MainFunction_Wakeup function performs the polling of wake-up events that are configured statically as 'to be polled'.	Not implemented for platform not support hardware wake-up.
SWS_Can_00 112	N/S	The function Can_MainFunction_Wakeup shall perform the polling of wake-up events that are configured statically as 'to be polled'.	Not implemented for platform not support hardware wake-up.
SWS_Can_00 185	N/S	The Can module may implement the function Can_MainFunction_Wakeup as empty define in case no polling at all is used.	AAI-279: reject this requirement because our SoC does not support wakeup.
SWS_Can_00 999	N/S	These requirements are not applicable to this specification.	this is not a requirement.
ECUC_Can_0 0357	N/S	Name CanMainFunctionWakeupPeriod Description This parameter describes the period for cyclic call to Can_MainFunction_Wakeup. Unit is seconds.	Shall be supported only by platforms which include hardware wake-up supported.
ECUC_Can_0 0430	N/S	Name CanSupportTTCANRef Description The parameter refers to CanIfSupportTTCAN parameter in the CAN Interface Module configuration. The CanIfSupportTTCAN parameter defines whether TTCAN is supported.	This is a HW limitation, some platform does not support TTCAN feature.

Table continues on the next page...

**Table 3-2. CAN Deviations Table (continued)**

Requirement	Status	Description	Notes
ECUC_Can_00466	N/S	Name CanWakeupFunctionalityAPI Description Adds / removes the service Can_CheckWakeup() from the code. True: Can_CheckWakeup can be used. False: Can_CheckWakeup cannot be used.	Shall be supported only by platforms which include hardware wake-up supported
ECUC_Can_00330	N/S	Name CanWakeupSupport Description CAN driver support for wakeup over CAN Bus.	Shall be supported only by platforms which include hardware wake-up supported. Always has value = false
ECUC_Can_00359	N/S	CanWakeupSourceRef 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). 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.	This requirement is only applicable for platform support HW wake up.

Can\_PBcfg\_VS.c files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB).

Can\_Cfg.c file will contain the definition for all parameters that are not variant aware.

### 3.5 Driver Limitation

- It is a hardware limitation that the FIFO feature must not be enabled when the CAN FD feature is enabled. It means the two features must not be used at the same time.
- The Can driver supports the reception FIFO engine whose size is fixed as 6-message deep due to the hardware limitation.
- The Can driver only supports ten the API name of Can\_MainFunction\_Write() for processing transmitted MBs. The APIs obey the following pattern: Can\_MainFunction\_Wrtte\_0(), Can\_MainFunction\_Write\_1(),



Can\_MainFunction\_Write\_2(), Can\_MainFunction\_Write\_3(), ... and until Can\_MainFunction\_Write\_10().

- The Can driver only supports ten the API name of Can\_MainFunction\_Read() for processing received MBs. The APIs obey the following pattern:  
Can\_MainFunction\_Read\_0(), Can\_MainFunction\_Read\_1(),  
Can\_MainFunction\_Read\_2(), Can\_MainFunction\_Read\_3(), ... and until  
Can\_MainFunction\_Read\_10().

## 3.6 Driver Usage and Configuration Tips

This chapter describes how to configure for advanced features which are not (fully) described by AUTOSAR SWS (i.e NON-ASR features).

### 3.6.1 Driver APIs usage

- Can\_AbortMb() API (Non Autosar) is defined if this feature is enabled by CanApiEnableMbAbort from the Tresos plugin. The call of the non-autosar function Can\_AbortMb shall be followed by the call of Can\_MainFunction\_Write when the pooling mode transmission is configured.
- Can\_SetClockMode() API (Non Autosar) is defined if the dual clock mode is enabled by the CanEnableDualClockMode node from the Tresos plugin.
- Multiplex transmission which is supported by Can\_Write() API means to send a message from any Tx MB that is free to be used, in the range of the same HWObjectID. This means that several Hardware Objects can have the same HWObjectID. This feature can be used only if it's enabled by CanMultiplexedTransmission from the Tresos Plugin. Refer to CAN277.
- Can driver support loop back mode to verify driver without hardware. In this case, it is necessary to configure filter in order to receive the transmit message.

### 3.6.2 Can bit timing configuration

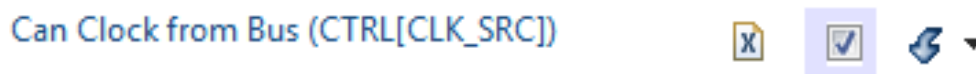
Each Can controller can be configured the Bit Time Interval (also known as Bit rate or Baudrate) individually by the CanControllerBaudrateConfig container from Tresos plugin. The CanControllerBaudrateConfig contains the elements where you can fill in the value for the bit timing variables of classical Can and the Can FD as well as the alternative bit timing variables (Can CBT).

This section provides you the method for determining and configuring the optimum bit timing parameters which satisfy the requirements for proper bit sampling.

### 3.6.2.1 Can bit time calculation

**Step 1:** Choose Can System Clock Frequency.

The clock source for CAN module can be obtained in 2 modes, depending by the setting of the **CanClockFromBus** control in Tresos.



**Figure 3-1. Can Clock Selection: bus**

If **CanClockFromBus** is checked, the clock value is extracted from peripheral clock. If **CanClockFromBus** is unchecked, the clock used by the hardware is extracted from external oscillator. It is user responsibility to check and to select the right clock source for their application.



**Figure 3-2. Can Clock Selection: osc\_clock**

Based on **CanClockFromBus** parameter, we have the frequency of CanController named **CanClockFrequency**(in Hertz).

**Step 2:** Determine the number of Time Quanta for the Can Bit time interval. The number of Time Quanta abbreviated as **No. of CanTimeQuantas** is an integer number from 8 to 25 and is calculated from the following equations

$$\text{TimeQuantum (seconds)} = \text{Prescaler} / \text{CanClockFrequency}$$

$$\text{No. of CanTimeQuantas} = (1 / \text{CancontrollerBaudRate}) / \text{TimeQuantum}$$

Where :

- **TimeQuantum** is the Can system periods in second.
- **Prescaler** which is the Prescaler division factor can be configured from the **CanControllerPrescaler** node in the **CanControllerBaudrateConfig** container.
- **CancontrollerBaudRate** which is the desire baudrate value can be configured from the **CancontrollerBaudRate** node in the **CanControllerBaudrateConfig** container.

You must choose the suitable value for Prescaler and CancontrollerBaudRate to obtain the No. of CanTimeQuantas inside the range from 8 to 25.

**Step 3:** Configure the integral number of Time Quanta for the time segments.

The bit time interval consists of three time segments, including SYNC\_SEG, TSEG1 and TSEG2, each specified as an exact integral number of Time Quanta.

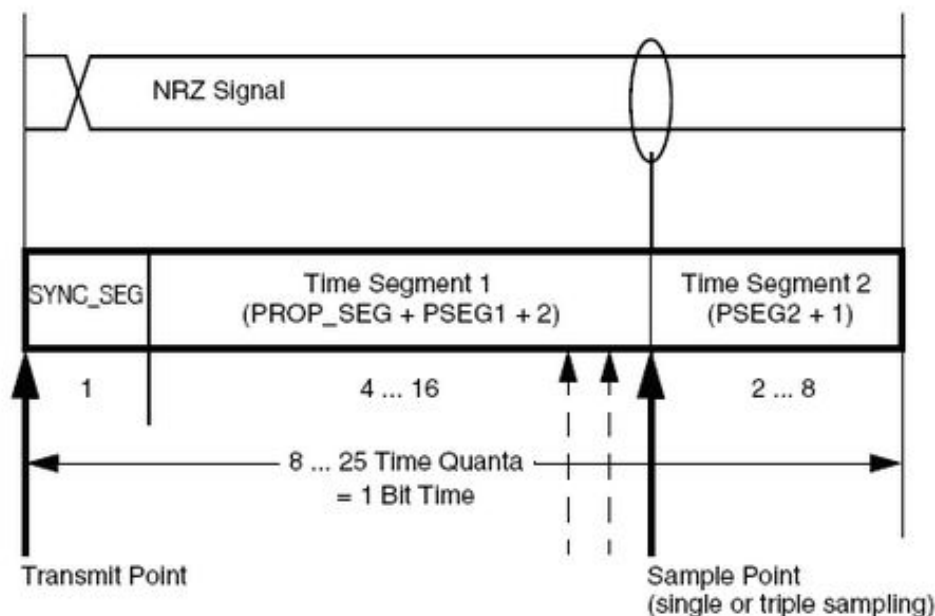
The segments are configured by four nodes from Tresos plugin, including: CanControllerSyncSeg, CanControllerPropSeg, CanControllerSeg1 and CanControllerSeg2. The relationship of the segments as following equation:

**No. of CanTimeQuantas = 1 + CanControllerPropSeg + CanControllerSeg1 + CanControllerSeg2**

- CanControllerSyncSeg defines Synchronization Segment or SYNC\_SEG. Its value is fixed as 1 because SYNC\_SEG has a fixed period of one Time Quanta
- CanControllerPropSeg and CanControllerSeg1 specify the number of Time Quanta of Time Segment 1 or TSEG1.
- CanControllerSeg2 specifies the number of Time Quanta of Time Segment 2 or TSEG2.

The sum of three segments must equal with **No. of CanTimeQuantas** resulted from Step 2. Thus, you can device the integral **No. of CanTimeQuantas** into three time segment as long as it is satisfied the requirement of each segment and the sample point recommendation.

The sample point is defined as following image. For the CAN protocol (J1939, J2284) is recommended to use a sample point in the range of 80% to 90%.



**Figure 3-3. Can Time Segments**

**Step 4:** Configure Re-synchronization Jump Width or SJW or RJW. RJW is chosen as the smaller of 4 and CanControllerSeg1. CanControllerSyncJumpWidth from Tresos plugin is used to configured for it.

### 3.6.2.2 Can FD bit time configuration

The Can FD protocol has 2 independent bit rates for the arbitration phase and the data phase. The arbitration phase uses the same bit timing as classical Can. The data bit rate is either the same or higher than the arbitration bit rate. The data bit rate of Can FD is configured by the CanControllerFdBaudrateConfig container from Tresos plugin.

The Can FD bit time configuration is the same as the above section (Can bit time calculation) within the parameters including: CanControllerFDPrescaller, CanControllerFdBaudRate, CanControllerFdSyncSeg, CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2, CanControllerSyncJumpWidth.

Can FD feature can not be used at the same time with the Rx Fifo feature due to the hardware limitation.

#### NOTE

When CAN FD is used, it is recommended to use Can CBT in configuring Can Bit Timing variables (See chapter Protocol timing in Reference Manual).

### 3.6.2.3 Can CBT usage and configuration

The Can CBT extends the range of the CAN bit timing variables in the Can bit time calculation section.

The Can CBT bit time configuration is the same as the above section (Can bit time calculation) within the parameters including: CanControllerCbtPrescaller, CanControllerCbtBaudRate, CanControllerCbtSyncSeg, CanControllerCbtPropSeg, CanControllerCbtSeg1, CanControllerCbtSeg2, CanControllerSyncJumpWidthCbt.

## 3.6.3 Can Hardware Object Handle (HOH) configuration

The CanHardwareObject container in Tresos plugin is used to configure the operating for Rx and Tx MBs. The elements of this container is detailed in the chapter [Can HardwareObject](#).

For Tx MBs (HTHs) the difference between Standard and Extended mode is done by the most significant bit of the Can ID. Refer to CANIF243 and CANIF188.

For Rx MBs (HRHs) the MIXED message buffer type is handled as EXTENDED type. The platform support only FIFO engine with 5 receive buffers storage scheme.

This section describes the configuration in the advanced features.

### 3.6.3.1 Configuration for Can FD feature

The Can driver supports to define the maximum payload allowed for the transmitted or received Can FD frames for each HOH when Can FD feature is enabled. In this feature, the FlexCAN RAM can be partitioned in blocks of 512 bytes, the payload size configuration is performed based on the configuration for RAM blocks.

When you want to select the same payload size for HOH, you do not need to care about the exact RAM block which the HOH is allocated in. RAM blocks will be configured the same payload size which depends on the configuration of the MBDSR parameter from Tresos plugin, so that HOH will be configured your desire payload size. In this case, you follow bellowing steps:

- Uncheck the CanSpecifiedRAMBlockSize node in Tresos plugin to disable the feature which allows to separately configure the HOH data size;
- Select the desire payload size in the MBDSR node.

In other case, you want to configure the different payload size for HOH, you must manage the message buffers allocation in RAM blocks because the different payload size is only obtained between the different blocks. The CanRAMBlockRef node in the CanHardwareObject container and the CanRAMBlock container from Tresos plugin support for this case. In this case, you follow bellowing steps:

- Check the CanSpecifiedRAMBlockSize node to enable the feature which allows to separately configure the HOH data size;
- Add elements for CanRAMBlock, each element is to represent for a RAM block configuration. In side the elements, you choose the RAM block at CanRAMBlockName and select the desire payload size at CanRAMBlockSizeValue;
- The CanRAMBlockRef node in the CanHardwareObject container will be editable and MBDSR will be changed to Gray. Enable CanRAMBlockRef then select an element of CanRAMBlock where you configured the payload size in.

### 3.6.3.2 HOH configuration for Rx FIFO feature

When this feature is enabled, it is required to allocate at least one HRH to the Can controller. The HRH which has the smallest CanObjectId in the list HRHs allocated to this controller is used to configure the operating of the FIFO structure (both Message Buffers and FIFO engine).

The CanHwFilter in the CanHardwareObject for Rx FIFO has no meaning in configuring the FIFO ID filter. The FIFO ID filter configuration will be performed in the CanRxFifo and CanRxFifoTable container.

See [Rx Fifo configuration](#) for more details.

### 3.6.4 Rx Fifo configuration

The receive-only FIFO is enabled for specific controller by asserting the FEN bit in the MCR register. The RxFifo configuration in the Tresos plugin is implemented by “CanControllerRxFifoEnable” parameter under “CanController” container.

When the Fifo is enabled, the memory region normally occupied by the first 6 MBs is normally reserved for use of the Fifo engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing the MB0 structure.

The interrupts corresponding to MB0 to 5 have a different behaviour when Rx Fifo is enabled. Bit 7 of the IFLAG1 becomes the “Fifo Overflow” flag, bit 6 becomes the “Fifo Warning” flag, bit 5 becomes the “Frame Available in Rx Fifo” flag and bits 4 to 0 are unused.

#### NOTE

The hardware objects configured in the “Can\_MBConfigObjectType” structure are used for the initialization of MBs. IF Rx Fifo is enabled then the MB initialization will start from the last MB which is not used by the RxFifo. The number of MB used by the RxFifo depends by the setting of CanRxFifoFiltersNumber parameter using the following formula:  $6 + (\text{CanRxFifoFiltersNumber} / 4)$ . As a result of this dependency the total number of hardware objects per controller is reduced to:  $32 - (6 + \text{CanRxFifoFiltersNumber} / 4)$  when Rx Fifo is enabled.

If RxFifo is enabled the user can define proper handlers for overflow and warnings notification events.

**NOTE**

If RxFifo is enabled for a specific controller, the user shall configure at least 1 hardware object which use that controller. The configuration container CanHwFilter has no meaning for this type of hardware object because the RxFifo has defined its own filtering id table. This is needed in order to access the Rxfifo using a hardware object handle, like an ordinary message buffer. In addition,

+ If CanBccSupport is disabled, then the CAN\_MCR[IRMQ] bit is negated and the CanMBFilterMaskValue in the container CanRxFifoTable is affected by CanRxFifoGlobalMaskValue. Rx matching process will be based on the masking value of CanRxFifoGlobalMaskValue;

+ If CanBccSupport is enabled, then the CAN\_MCR[IRMQ] bit is asserted and Rx matching process will be based on the masking value configured by the CanMBFilterMaskValue in the container CanRxFifoTable. The CanRxFifoGlobalMaskValue parameter does not affect to the Rx FIFO ID filter.

Below is presented an example of a mapping between hardware objects and message buffers for a driver configuration which use multiple controllers and the RxFifo feature is enabled for all of them:

HRH0 id 0, controller A -> rx fifo of controller A

HRH1 id 1, controller A -> MB8

HRH2 id 2, controller A -> MB9

HRH3 id 3, controller B -> rx fifo of controller B

HRH4 id 4, controller B -> MB8

HRH5 id 5, controller B -> MB9

HTH0 id 6, controller A -> MB10

HTH1 id 7, controller B -> MB10

In order to understand the differences, below is presented an example of a mapping between hardware objects and message buffers for a driver configuration which use multiple controllers and the RxFifo feature is NOT enabled for any controller:

HRH0 id 0, controller A -> MB0

HRH1 id 1, controller A -> MB1

HRH2 id 2, controller A -> MB2

HRH3 id 3, controller B -> MB0

HRH4 id 4, controller B -> MB1

HRH5 id 5, controller B -> MB2

HTH0 id 6, controller A -> MB3

HTH1 id 7, controller B -> MB4

## 3.7 Runtime Errors

The driver generates the following DEM errors at runtime.

**Table 3-3. Runtime Errors**

Function	Error Code	Condition triggering the error
All functions	CAN_E_UNINIT	The driver is not yet initialized
Can_Flexcan_Write	CAN_E_PARAM_DATALENGHT	The length is more than 8 bytes
Mcu_SetMode	CAN_E_PARAM_CONTROLLER	The parameter Controller is out of range
Can_FlexCan_ProcessRxNormal	CAN_E_DATALOST	In case of OVERWRITE or OVERRUN event detection
Can_Flexcan_ProcessRxFIFO	CAN_E_DATALOST	mbindex corresponds to flags for Int FiFo status: Overflow or Warning
Can_Init and Can_Flexcan_UpdateMB	CAN_E_PARAM_POINTER	The parameter PduInfo or the SDU pointer inside PduInfo is a null-pointer
Can_GetVersionInfo	CAN_E_PARAM_POINTER	The paramete versionInfo is a null pointer
Can_Init, Can_ChangeBaudrate, Can_SetBaudRate	CAN_E_TRANSITION	The controller is not in CANIF_CS_STOPPED
Can_FlexCan_SetcontrollerToSleepMode, Can_FlexCan_SetcontrollerToStart Mode	CAN_E_TRANSITION	The controller is not in STOP state
Can_FlexCan_SetcontrollerToWakeUpMode	CAN_E_TRANSITION	The controller is not in SLEEP state
Can_FlexCan_SetcontrollerMode	CAN_E_TRANSITION	A invalid transition has been requested
Can_FlexCan_SetcontrollerToStop Mode	CAN_E_TRANSITION	The controller is not in START or WAKEUP state



## 3.8 Software specification

The following sections contains driver software specifications.

### 3.8.1 Define Reference

Constants supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.3 Rev0001 .

#### 3.8.1.1 Define CAN\_API\_ENABLE\_ABORT\_MB

Support for Special MB Abort API.

**Definition:** `#define CAN_API_ENABLE_ABORT_MB (STD_ON)`

#### 3.8.1.2 Define CAN\_BCC\_SUPPORT\_ENABLE

Defines if Backwards Compatibility Configuration (BCC) feature of CAN controller is used in the configuration. If BCC feature of CAN controller is enabled, Individual Rx masking and queue feature are disabled. If BCC feature of CAN controller is disabled, Individual Rx masking and queue feature are enabled.

**Definition:** `#define CAN_BCC_SUPPORT_ENABLE (STD_OFF)`

#### 3.8.1.3 Define CAN\_BUSOFFPOLL\_SUPPORTED

This macro enables `Can_MainFunction_BusOff()` if at least one controller is set to process BusOff in Polling Mode.

**Definition:** `#define CAN_BUSOFFPOLL_SUPPORTED (STD_ON)`

#### 3.8.1.4 Define CAN\_SET\_BAUDRATE\_API

This macro switches the `Can_SetBaudrate` API and `Can_SetBaudRate` API ON or OFF.

**Definition:** `#define CAN_SET_BAUDRATE_API (STD_ON)`

### 3.8.1.5 Define CAN\_DEV\_ERROR\_DETECT

Switches the Development Error Detection and Notification ON or OFF.

**Definition:** `#define CAN_DEV_ERROR_DETECT (STD_ON)`

### 3.8.1.6 Define CAN\_DUAL\_CLOCK\_MODE

Enable Non-Autosar API for Dual-Clock support.

**Definition:** `#define CAN_DUAL_CLOCK_MODE (STD_OFF)`

### 3.8.1.7 Define CAN\_E\_DEFAULT

Development errors.

**Definition:** `#define CAN_E_DEFAULT (uint8)0x08U`

**Detail:**

This feature is reserved for future

### 3.8.1.8 Define CAN\_E\_DATAHOST

Development errors.

**Definition:** `#define CAN_E_DATAHOST (uint8)0x01U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportRuntimeError function.

**Condition:**

This event will occur whenever have no data in MB.

**Suggestion:**

If the development error detection for the Can module is enabled, the Can module shall raise the error CAN\_E\_DATAHOST in case of OVERWRITE or OVERRUN event detection.

### 3.8.1.9 Define CAN\_E\_PARAM\_CONTROLLER

Development errors.

**Definition:** `#define CAN_E_PARAM_CONTROLLER (uint8)0x04U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

This event will occur if the parameter Controller is out of range.

**Suggestion:**

Check the parameter Controller.

### 3.8.1.10 Define CAN\_E\_PARAM\_DATA\_LENGTH

Development errors.

**Definition:** `#define CAN_E_PARAM_DATA_LENGTH (uint8)0x03U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

The function Can\_Write shall raise the error CAN\_E\_PARAM\_DATA\_LENGTH if the length is more than 8 byte.

**Suggestion:**

Check the PDU length, it must smaller than 8 byte.

### 3.8.1.11 Define CAN\_E\_PARAM\_HANDLE

Development errors.

**Definition:** `#define CAN_E_PARAM_HANDLE (uint8)0x02U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

The function Can\_Write shall raise the error CAN\_E\_PARAM\_HANDLE if the parameter Hth is not a configured Hardware Transmit Handle.

### 3.8.1.12 Define CAN\_E\_PARAM\_POINTER

Development errors.

**Definition:** `#define CAN_E_PARAM_POINTER (uint8)0x01U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

This event will occur in some function when the pointer is null.

**Suggestion:**

The function Can\_GetVersionInfo shall raise the error CAN\_E\_PARAM\_POINTER if the parameter versionInfo is a null pointer.

The function Can\_Init shall raise the error CAN\_E\_PARAM\_POINTER if a NULL pointer was given as config parameter.

The function Can\_Write shall raise the error CAN\_E\_PARAM\_POINTER if the parameter PduInfo or the SDU pointer inside PduInfo is a null-pointer and return CAN\_NOT\_OK.

### 3.8.1.13 Define CAN\_E\_TRANSITION

Development errors.

**Definition:** `#define CAN_E_TRANSITION (uint8)0x06U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

This event will occur whenever the driver state transition is invalid.

**Suggestion:**

If an invalid transition has been requested, function shall raise the error CAN\_E\_TRANSITION and return CAN\_NOT\_OK.

### 3.8.1.14 Define CAN\_E\_UNINIT

Development errors.

**Definition:** `#define CAN_E_UNINIT (uint8)0x05U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

This event will occur if the driver is not yet initialized.

**Suggestion:**

In many functions need check the init of Can module. If the module is not yet initialized then return CAN\_NOT\_OK

### 3.8.1.15 Define CAN\_E\_PARAM\_BAUDRATE

Development errors.

**Definition:** `#define CAN_E_PARAM_BAUDRATE (uint8)0x07U`

**Violates:** MISRA 2004 Required Rule 16.10, Don't check return type of Det\_ReportError function.

**Condition:**

This event will occur if the parameter Baudrate has an invalid value.

**Suggestion:**

Check the parameter Baudrate.

### 3.8.1.16 Define CAN\_ERROR\_NOTIFICATION\_ENABLE

Error notification enabled/disabled.

**Definition:** `#define CAN_ERROR_NOTIFICATION_ENABLE (STD_ON)`

### 3.8.1.17 Define CAN\_EXTENDEDID

Extended identifiers.

**Definition:** `#define CAN_EXTENDEDID (STD_ON)`

### 3.8.1.18 Define CAN\_INIT\_CONFIG\_CONTROLLERS\_PC

**Definition:** `#define CAN_INIT_CONFIG_CONTROLLERS_PC extern  
CONST(Can_ControllerBaudrateConfigType, CAN_CONST)  
ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT_0];`

### 3.8.1.19 Define CAN\_INIT\_CONFIG\_PC\_DEFINES

Export Can\_ConfigType structure.

**Definition:** `#define CAN_INIT_CONFIG_PC_DEFINES extern CONST(Can_ConfigType, CAN_CONST)  
Can_ConfigSet_PC;`

### 3.8.1.20 Define CAN\_ISROPTCODESIZE

Optimization of interrupt service code for size.

**Definition:** `#define CAN_ISROPTCODESIZE (STD_OFF)`

### 3.8.1.21 Define CAN\_MAINFUNCTION\_MODE\_PERIOD

Periods for cyclic call of Main function Mode.

**Definition:** `#define CAN_MAINFUNCTION_MODE_PERIOD 0U`

### 3.8.1.22 Define CAN\_MAINFUNCTION\_PERIOD\_BUSOFF

Periods for cyclic call of Main function.

**Definition:** `#define CAN_MAINFUNCTION_PERIOD_BUSOFF 0U`

### 3.8.1.23 Define CAN\_MAINFUNCTION\_PERIOD\_READ

Periods for cyclic call of Main function.

**Definition:** `#define CAN_MAINFUNCTION_PERIOD_READ 1U`

### 3.8.1.24 Define CAN\_MAINFUNCTION\_PERIOD\_WRITE

Periods for cyclic call of Main function Write.

**Definition:** `#define CAN_MAINFUNCTION_PERIOD_WRITE 3U`

### 3.8.1.25 Define CAN\_MAXCTRL\_SUPPORTED

Maximum possible controllers per specific derivative.

**Definition:** `#define CAN_MAXCTRL_SUPPORTED 2U`

### 3.8.1.26 Define CAN\_MAXMB\_SUPPORTED

Maximum possible Message Buffers per controller specific to this platform.

**Definition:** `#define CAN_MAXMB_SUPPORTED 32U`

### 3.8.1.27 Define CAN\_MBCOUNTTEXTENSION

Extended number of can hardware objects.

**Definition:** `#define CAN_MBCOUNTTEXTENSION (STD_ON)`

### 3.8.1.28 Define CAN\_MIX\_MB\_SUPPORT

Platform support mix of controllers with 64 and 32 MBs.

**Definition:** `#define CAN_MIX_MB_SUPPORT (STD_OFF)`

### 3.8.1.29 Define CAN\_MULTIPLEXED\_TRANSMISSION

Support for Multiplexed Transmission.

**Definition:** `#define CAN_MULTIPLEXED_TRANSMISSION (STD_ON)`

### 3.8.1.30 Define CAN\_PRECOMPILE\_SUPPORT

Precompile Support On.

**Definition:** `#define CAN_PRECOMPILE_SUPPORT`

### 3.8.1.31 Define CAN\_RXFIFO\_ENABLE

Support for Rx Fifo.

**Definition:** `#define CAN_RXFIFO_ENABLE (STD_ON)`

### 3.8.1.32 Define CAN\_RXFIFO\_EVENT\_UNIFIED

Set if Rx Fifo events (Warning/Overflow/FrameAvailable) are configured on the same int on INTC vector table.

**Definition:** `#define CAN_RXFIFO_EVENT_UNIFIED (STD_ON)`

### 3.8.1.33 Define CAN\_RXPOLL\_SUPPORTED

This macro enables `Can_MainFunction_Read()` if at least one controller is set to process Rx in Polling Mode.

**Definition:** `#define CAN_RXPOLL_SUPPORTED (STD_ON)`

### 3.8.1.34 Define CAN\_SID\_ABORT\_MB

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_ABORT_MB (uint8)0x10U`

### 3.8.1.35 Define CAN\_SID\_DISABLE\_CONTROLLER\_INTERRUPTS

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_DISABLE_CONTROLLER_INTERRUPTS (uint8)0x04U`

### 3.8.1.36 Define CAN\_SID\_ENABLE\_CONTROLLER\_INTERRUPTS

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_ENABLE_CONTROLLER_INTERRUPTS (uint8)0x05U`

### 3.8.1.37 Define CAN\_SID\_GET\_VERSION\_INFO

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_GET_VERSION_INFO (uint8)0x07U`



### 3.8.1.38 Define CAN\_SID\_INIT

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_INIT (uint8) 0x00U`

### 3.8.1.39 Define CAN\_SID\_MAIN\_FUNCTION\_BUS\_OFF

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_MAIN_FUNCTION_BUS_OFF (uint8) 0x09U`

### 3.8.1.40 Define CAN\_SID\_MAIN\_FUNCTION\_MODE

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_MAIN_FUNCTION_MODE (uint8) 0x0CU`

### 3.8.1.41 Define CAN\_SID\_MAIN\_FUNCTION\_READ

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_MAIN_FUNCTION_READ (uint8) 0x08U`

### 3.8.1.42 Define CAN\_SID\_MAIN\_FUNCTION\_WAKEUP

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_MAIN_FUNCTION_WAKEUP (uint8) 0x0AU`

### 3.8.1.43 Define CAN\_SID\_MAIN\_FUNCTION\_WRITE

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_MAIN_FUNCTION_WRITE (uint8) 0x01U`

### 3.8.1.44 Define CAN\_SID\_SET\_CONTROLLER\_MODE

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_SET_CONTROLLER_MODE (uint8)0x03U`

### 3.8.1.45 Define CAN\_SID\_SETCLOCKMODE

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_SETCLOCKMODE (uint8)0x0FU`

### 3.8.1.46 Define CAN\_SID\_WRITE

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_WRITE (uint8)0x06U`

### 3.8.1.47 Define CAN\_SID\_SET\_BAUDRATE

Service ID (APIs) for Det reporting.

**Definition:** `#define CAN_SID_SET_BAUDRATE (uint8)0x0FU`

### 3.8.1.48 Define CAN\_SET\_BAUDRATE\_API

This macro switches the Can\_SetBaudrate API and Can\_SetBaudRate API ON or OFF.

**Definition:** `#define CAN_SET_BAUDRATE_API (STD_ON)`

### 3.8.1.49 Define CAN\_TIMEOUT\_DURATION

(CAN113\_Conf) Specifies the maximum time for blocking function until a timeout is detected. Unit in loops.

**Definition:** `#define CAN_TIMEOUT_DURATION 20U`

### 3.8.1.50 Define CAN\_TXPOLL\_SUPPORTED

This macro enables Can\_MainFunction\_Write() if at least one controller is set to process Tx in Polling Mode.

**Definition:** `#define CAN_TXPOLL_SUPPORTED (STD_ON)`

### 3.8.1.51 Define CAN\_VERSION\_INFO\_API

Support for version info API.

**Definition:** `#define CAN_VERSION_INFO_API (STD_ON)`

## 3.8.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.3 Rev0001 .

### 3.8.2.1 Enumeration Can\_ClockModeType

CAN source clock selection used in Can\_SetClockMode Non-Autosar API.

**Table 3-4. Enumeration Can\_ClockModeType Values**

Value	Description
CAN_NORMAL = 0U	Standard configuration (default).
CAN_ALTERNATE	Second configuration (special).

### 3.8.2.2 Enumeration Can\_ControllerStateType

States that defines the controllers.

**Table 3-5. Enumeration Can\_ControllerStateType Values**

Value	Description
CAN_STOPPED = 0U	Controller in state STOPPED.
CAN_STARTED	Controller in state STARTED.
CAN_SLEEPS	Controller in state SLEEPED.

### 3.8.2.3 Enumeration Can\_ObjType

Used for value received by Tressos interface configuration. Describe the MB configuration.

**Table 3-6. Enumeration Can\_ObjType Values**

Value	Description
CAN_RECEIVE = 0U	Receive MB.
CAN_TRANSMIT	Transmit MB.

### 3.8.2.4 Enumeration Can\_ReturnType

CAN Return Types from Functions.

**Table 3-7. Enumeration Can\_ReturnType Values**

Value	Description
CAN_OK = 0U	Operation was ok executed.
CAN_NOT_OK	Operation was not ok executed.
CAN_BUSY	Operation was rejected because of busy state.

### 3.8.2.5 Enumeration Can\_ControllerStateType

State transitions that are used by the function CAN\_SetControllerMode().

**Table 3-8. Enumeration Can\_StateTransitionType Values**

Value	Description
CAN_CS_STOPPED = 0U	CANIF_CS_STARTED -> CANIF_CS_STOPPED.
CAN_CS_STARTED	CANIF_CS_STOPPED -> CANIF_CS_STARTED.
CAN_CS_SLEEP	CANIF_CS_STOPPED -> CANIF_CS_SLEEP.

### 3.8.2.6 Enumeration Can\_StatusType

CAN Driver status used for checking and preventing double driver initialization.

CAN\_UNINIT = The CAN controller is not initialized. The CAN Controller is not participating on the CAN bus. All registers belonging to the CAN module are in reset

state, CAN interrupts are disabled. CAN\_READY = Controller has initialized: static variables, including flags; Common setting for the complete CAN HW unit; CAN controller specific settings for each CAN controller.

**Table 3-9. Enumeration Can\_StatusType Values**

Value	Description
CAN_UNINIT = 0U	Driver not initialized.
CAN_READY	Driver ready.

### 3.8.2.7 Enumeration CanIdType

Used for value received by Tressos interface configuration. Used to differentiate Extended, Mixed or Standard Id type

**Table 3-10. Enumeration CanIdType Values**

Value	Description
CAN_EXTENDED = 0U	Extended ID (29 bits).
CAN_STANDARD	Standard ID (11 bits).
CAN_MIXED	Mixed ID (29 bits).

### 3.8.2.8 Enumeration Can\_FdType

Can\_FdType.

Used for value received by Tressos interface configuration.

**Table 3-11. Enumeration Can\_FdType Values**

Value	Description
CAN_8_BYTES_PAYLOAD = 0U	Message buffer Data size.
CAN_16_BYTES_PAYLOAD	Message buffer Data size.
CAN_32_BYTES_PAYLOAD	Message buffer Data size.
CAN_64_BYTES_PAYLOAD	Message buffer Data size.

## 3.8.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.3 Rev0001 .

### 3.8.3.1 Function Can\_Init

Initialize the CAN driver. SID = 0x00.

**Prototype:** void Can\_Init(const Can\_ConfigType \*Config);

**Table 3-12. Can\_Init Arguments**

Type	Name	Direction	Description
const Can_ConfigType *	Config	input	Pointer to driver configuration.

**Return:** void

Initialize all the controllers. The CAN module shall be initialized by Can\_Init(<&Can\_Configuration>) service call during the start-up. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Can\_Init shall be called at most once during runtime.

**post:** Can\_Init shall initialize all the controllers and set the driver in READY state.

### 3.8.3.2 Function Can\_GetVersionInfo

Returns the version information of this module. SID = 0x07.

**Prototype:** void Can\_GetVersionInfo(Std\_VersionInfoType \*versioninfo);

**Table 3-13. Can\_GetVersionInfo Arguments**

Type	Name	Direction	Description
Std_VersionInfoType *	versioninfo	input	A pointer to location to store version info Must be omitted if the function does not have parameters.

**Return:** void

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** The CAN\_VERSION\_INFO\_API define must be configured on.

**post:** The version information is return if the parameter versionInfo is not a null pointer.

### 3.8.3.3 Function Can\_SetControllerMode

Put the controller into a required state. SID = 0x03.

**Prototype:** `Can_ReturnType Can_SetControllerMode(uint8 Controller, Can_StateTransitionType Transition);`

**Table 3-14. Can\_SetControllerMode Arguments**

Type	Name	Direction	Description
uint8	Controller	input	- Can controller for which the status shall be changed - based on configuration order list (CanControllerId).
Can_StateTransitionType	Transition	input	- Possible transitions (CAN_T_STOP / CAN_T_START / CAN_T_SLEEP / CAN_T_WAKEUP)

**Return:** Can\_ReturnType Result of the transition.

**Table 3-15. Can\_SetControllerMode Returns**

Value	Description
CAN_OK	Transition initiated.
CAN_NOT_OK	Development or production error.

Switch the controller from one state to another. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Before changing the controller state the driver must be initialized.

**post:** After the transition to the new state the interrupts required for that state must be enabled.

### 3.8.3.4 Function Can\_DisableControllerInterrupts

Disable INTs. SID = 0x04.

**Prototype:** `void Can_DisableControllerInterrupts(uint8 Controller);`

**Table 3-16. Can\_DisableControllerInterrupts Arguments**

Type	Name	Direction	Description
uint8	Controller	input	Can controller for which interrupts shall be disabled - based on configuration order list (CanControllerId).

**Return:** void

Switch OFF the controller's interrupts. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized before changing the interrupts state (en/dis).

**post:** Controller must not respond to any interrupt assertion.

#### Note

The maximum number of nested calls to Can\_DisableControllerInterrupts is limited to 127. This function may not be preempted by code which calls function Can\_InitController

### 3.8.3.5 Function Can\_EnableControllerInterrupts

Enable INTs. SID = 0x05.

**Prototype:** void Can\_EnableControllerInterrupts(uint8 Controller);

**Table 3-17. Can\_EnableControllerInterrupts Arguments**

Type	Name	Direction	Description
uint8	Controller	input	Can controller for which interrupts shall be disabled - based on configuration order list (CanControllerId).

**Return:** void

Switch ON the controller's interrupts. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized before changing the interrupts state (en/dis).

**post:** Controller must respond to interrupt assertion.

#### Note

This function may not be preempted by code which calls function Can\_InitController.

### 3.8.3.6 Function Can\_Write

Transmit information on CAN bus. SID = 0x06.



**Prototype:** `Can_ReturnType Can_Write(Can_HwHandleType Hth, const Can_PduType *PduInfo);`

**Table 3-18. Can\_Write Arguments**

Type	Name	Direction	Description
Can_HwHandleType	Hth	input	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.
const Can_PduType *	PduInfo	input	Pointer to SDU user memory, DLC and Identifier.

**Return:** Can\_ReturnType Result of the write operation.

**Table 3-19. Can\_Write Returns**

Value	Description
CAN_OK	Write command has been accepted.
CAN_NOT_OK	Development error occurred.
CAN_BUSY	No of TX hardware buffer available or preemptive call of Can_Write() that can't be implemented reentrant.

Can\_Write checks if hardware transmit object that is identified by the HTH is free. Can\_Write checks if another Can\_Write is ongoing for the same HTH. a) hardware transmit object is free: The mutex for that HTH is set to 'signaled' the ID, DLC 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 CAN\_OK. b) hardware transmit object is busy with another transmit request. The function returns with CAN\_BUSY. c) A preemptive call of Can\_Write has been issued, that could not be handled reentrant (i.e. a call with the same HTH). The function returns with CAN\_BUSY the function is non blocking d) The hardware transmit object is busy with another transmit request for an L-PDU that has lower priority than that for the current request The transmission of the previous L-PDU is cancelled (asynchronously). The function returns with CAN\_BUSY. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized and MB must be configured for Tx.

**post:** The data can be transmitted or rejected because of another data with a higher priority.

### 3.8.3.7 Function Can\_MainFunction\_Write

Function called at fixed cyclic time. SID 0x01.

**Prototype:** `void Can_MainFunction_Write(void);`

Service for performs the polling of TX confirmation and TX cancellation confirmation when CAN\_TX\_PROCESSING is set to POLLING. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized.

**post:** Send the data from that MB that is configured for Tx.

### 3.8.3.8 Function Can\_MainFunction\_Read

Function called at fixed cyclic time, with polling on only one MessageBuffer.

**Prototype:** `void Can_MainFunction_Read(void);`

Service for performs the polling of RX indications when CAN\_RX\_PROCESSING is set to POLLING. This routine is called by:

**pre:** Driver must be initialized.

**post:** Receive the data from that MB that is configured for Rx.

#### Note

This function may not be preempted by code which calls any of the driver functions.

### 3.8.3.9 Function Can\_MainFunction\_BusOff

Function called at fixed cyclic time, with polling on only one MessageBuffer.

**Prototype:** `void Can_MainFunction_BusOff(void);`

Service for performs the polling of BusOff events that are configured statically as 'to be polled'. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized.

**post:** Handle the Busoff event.

### 3.8.3.10 Function Can\_MainFunction\_Mode

Function called at fixed cyclic time. SID = 0x0C.

**Prototype:** `void Can_MainFunction_Mode(void);`

Service for performs performs the polling of CAN status register flags to detect transition of CAN Controller state This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized.

**post:** Handle the transition of Can Controller state.

### 3.8.3.11 Function Can\_AbortMb

Process a message buffer abort.

**Prototype:** `void Can_AbortMb(Can_HwHandleType Hth);`

**Table 3-20. Can\_AbortMb Arguments**

Type	Name	Direction	Description
Can_HwHandleType	Hth	input	- HW-transmit handler

This function write a abort code (b'1001) to MBCS[CODE] field of the MB. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre::** Driver must be initialized and the current MB transmission should be ready for transmit.

### 3.8.3.12 Function Can\_SetClockMode

Process a transition from one clock source to another.

**Prototype:** `Std_ReturnType Can_SetClockMode(uint8 can_controller, Can_ClockModeType can_clk_mode);`

**Table 3-21. Can\_SetClockMode Arguments**

Type	Name	Direction	Description
uint8	can_controller	input	controller ID
Can_ClockModeType	can_clk_mode	input	clock mode selection

**Return:** Std\_ReturnType Result of the clock switch operation.

**Table 3-22. Can\_SetClockMode Returns**

Value	Description
E_OK	Switch clock operation was ok.
E_NOT_OK	Switch clock operation was not ok.

This function is configuring Can controllers to run on the same baudrate, but having a different MCU source clock. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Driver must be initialized and all the controllers must be in Stop state.

### 3.8.3.13 Function Can\_SetBaudrate

Initialize the CAN controllers. SID = 0x0f.

**Prototype:** Std\_ReturnType Can\_SetBaudrate( uint8 Controller, uint16 BaudRateConfigID )

**Table 3-23. Can\_SetBaudrate Arguments**

Type	Name	Direction	Description
uint8	Controller	input	CAN Controller, whose baudrate shall be changed
uint16	BaudRateConfigID	input	references a baud rate configuration by ID

**Return:** Std\_ReturnType

Initialize all the controllers. Initialize the controller based on ID input parameter. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre::** Before controller re-initialization the driver must be initialized and the controllers must be in Stop state.

**post::** Interrupts and MBs must be configured for respond to CAN bus.

### 3.8.3.14 Function Can\_DeInit

Initialize the CAN driver. SID = 0x10

**Prototype:** `void Can_Deinit (void);`

**Table 3-24. Can\_DeInit Arguments**

Type	Name	Direction	Description

**Return:** void.

DeInitialize all the controllers. The CAN module shall be initialized by Can\_DeInit() service call during the start-up. This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

**pre:** Can\_DeInit shall be called at most once during runtime.

**post:** Can\_DeInit shall de initialize all the controllers and set the driver in UNINIT state.

### 3.8.3.15 Function Can\_GetControllerMode

Initialize the CAN driver. SID = 0x12.

**Prototype:** `Std_ReturnType Can_GetControllerMode(uint8 Controller, Can_ControllerStateType* ControllerModePtr);`

**Table 3-25. Can\_Write Arguments**

Type	Name	Direction	Description
uint8	Controller	input	ACAN controller for which the status shall be requested
const Can_PduType *	ControllerModePtr	output	Pointer to a memory location, where the current mode of the CAN controller will be stored

**Return:** Std\_ReturnType.

The CAN module shall be Get Controller Mode by Can\_GetControllerMode() service call during the start-up. This routine is called by: • CanIf or an upper layer according to Autosar requirements

- CanIf or an upper layer according to Autosar requirements.

**pre:****post:**

### 3.8.3.16 Function Can\_GetControllerErrorState

Initialize the CAN driver. SID = 0x11.

**Prototype:** Std\_ReturnType Can\_GetControllerErrorState(uint8 ControllerId, Can\_ErrorStateType\* ErrorStatePtr);

**Table 3-26. Can\_Write Arguments**

Type	Name	Direction	Description
uint8	ControllerId	input	Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState
Can_ErrorStateType*	ErrorStatePtr	output	Pointer to a memory location, where the error state of the CAN controller will be stored

**Return:** Std\_ReturnType

The CAN module shall be Get Controller Error State by Can\_GetControllerErrorState() service call during the start-up. This routine is called by: • CanIf or an upper layer according to Autosar requirements

- CanIf or an upper layer according to Autosar requirements.

**pre:****post:**

## 3.8.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.3 Rev0001 .

### 3.8.4.1 Structure Can\_ConfigType and CanStatic\_ConfigType

Top Level structure containing all Driver configuration.

A pointer to this structure is transmitted to `Can_Init()` to initialize the driver at startup. The application selects one of the configurations by using a pointer to one of the elements of this array as a parameter of the `Can_Init` function.

## Declaration

```
typedef struct
{
    Can_IdPtrType pFilterMasks,
    const Can_MBConfigContainerType MBConfigContainer,
    const Can_ControllerDescriptorType *ControllerDescriptors,
    Can_RxFifoTableIdConfigType *pRxFifoTableIdConfig,
    const Can_RxFilterTableType pRxFilterTableConfig,
    uint32 u32CanMaxObjectId,
    const Can_HwHandleType uCanFirstHTHIndex,
} Can_ConfigType;
```

**Table 3-27. Structure Can\_ConfigType member description**

Member	Description
pFilterMasks	Pointer to the first FilterMask value - any controller can have many filter masks for Can messages.
MBConfigContainer	Pointer to the first MB configuration of this Controller.
ControllerDescriptors	Pointer to the first CAN Controller description.
pRxFifoTableIdConfig	Pointer to the Table IDs for the RxFifo.
pRxFilterTableConfig	The Rx Filter Table.
u32CanMaxObjectId	Maximum Object IDs configured.
uCanFirstHTHIndex	The index of the first HTH configured.

```
typedef struct
{
    uint8 u8ControllersConfigured,
    CanStatic_ControllerDescriptorType *StaticControllerDescriptors
} CanStatic_ConfigType;
```

**Table 3-28. Structure CanStatic\_ConfigType member description**

Member	Description
u8ControllersConfigured	Number of Can Controllers configured in Tresos plugin.
StaticControllerDescriptors	Pointer to the first FlexCAN Controller description.

### 3.8.4.2 Structure Can\_ControllerDescriptorType

Structure for describing individual CAN controllers on the chip.

HRH = Hardware Receive Handle (HRH) is defined and provided by the CAN driver. Each HRH represents exactly one hardware object. The HRH can be used to optimize software filtering. HTH = The Hardware Transmit Handle (HTH) is defined and provided by the CAN driver. Each HTH represents one or several hardware objects, that are configured as hardware transmit pool.

## Declaration

## Software specification

```
typedef struct
{
    const uint8 u8MaxBaudRateCount,
    const uint8 u8DefaultBaudRateIndex,
    const Can_ControllerBaudrateConfigType *pControllerBaudrateConfigsPtr,
    const uint32 *u32RxFifoGlobalMask,
    const uint8 *u8RxFifoUsedMb,
    const Can_PCallbackType Can_RxFifoOverflowNotification,
    const Can_PCallbackType Can_RxFifoWarningNotification,
    const uint32 u32Options,
} Can_ControllerDescriptorType;
```

**Table 3-29. Structure Can\_ControllerDescriptorType member description**

Member	Description
u8MaxBaudRateCount	Max BaudRate number.
u8DefaultBaudRateIndex	Default baudrate index.
pControllerBaudrateConfigsPtr	Pointer to the Configuration of Baudrate timing parameter for FlexCAN baudrate controller ( CTRL value register).
u32RxFifoGlobalMask	Rx Fifo Global mask value
u8RxFifoUsedMb	Number of MBs used by Rx Fifo
Can_RxFifoOverflowNotification	Pointer to RX FIFO Overflow notification function.
Can_RxFifoWarningNotification	Pointer to RX FIFO Warning notification function.
u32Options	BusOff Sw Recovery, RXFifo En, IDAM Type, Event Trigger Mode TxProcessing/RxProcessing/BusoffProcessing/WakeUpProcessing: Polling vs Interrupt mode.

```
typedef struct
{
    const uint8 u8ControllerOffset,
    const uint8 u8MaxMBCount,
    const CanStatic_ControllerBaudrateConfigType *pStaticControllerBaudrateConfigsPtr,
    const Can_PCallbackType Can_ErrorNotification,
    const uint8 u8NumberOfMB,
} CanStatic_ControllerDescriptorType;
```

**Table 3-30. Structure CanStatic\_ControllerDescriptorType member description**

Member	Description
u8ControllerOffset	Hardware Offset for Can controller: FLEXCAN_A = Offset[0], FLEXCAN_B = Offset[1], ...
u8MaxMbTxCount	Rx Fifo Global mask value
pStaticControllerBaudrateConfigsPtr	Pointer to the Configuration of Baudrate timing parameter for FlexCAN baudrate controller ( CTRL value register).
Can_ErrorNotification	Pointer to Error interrupt notification function (ESR[ERR_INT]).
u8NumberOfMB	Number of message Buffers available for FlexCan unit.

### 3.8.4.3 Structure Can\_ControllerBaudrateConfigType and CanStatic\_ControllerBaudrateConfigType

Configuration of CAN controller.



This structure is initialized by Tresos considering user settings. Used by `Can_ConfigType` and `CanStatic_ConfigType`. Passed to `Can_InitController()` at initialization.

## Declaration

```
typedef struct
{
    const uint32 u32ControlRegister,
    const uint8 u8TxArbitrationStartDelay;
    const uint32 u32ControlRegisterAlternate,
    const uint16 u16ControllerBaudRate,
    const Can_ControllerFdConfigType ControllerFD,
    const Can_ControllerCbtConfigType ControllerCbtRegister,
    const uint16 u16ControllerBaudRateConfigID,
} Can_ControllerBaudrateConfigType;
```

**Table 3-31. Structure Can\_ControllerBaudrateConfigType member description**

Member	Description
u32ControlRegister	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
u8TxArbitrationStartDelay	The value of the Tx Arbitration Start Delay (TASD) bit field.
u32ControlRegisterAlternate	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
u16ControllerBaudRate	Configured BaudRate in kbps.
ControllerFD	Content of the CANFD_CBT Register fields.
ControllerCbtRegister	Content of the CAN_CBT Register fields.
u16ControllerBaudRateConfigID	The ID of Controller baudrate configuration.

```
typedef struct
{
    const uint32 u32ControlRegister,
    const uint32 u32ControlRegisterAlternate,
} CanStatic_ControllerBaudrateConfigType;
```

**Table 3-32. Structure CanStatic\_ControllerBaudrateConfigType member description**

Member	Description
u32ControlRegister	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
u32ControlRegisterAlternate	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.

### 3.8.4.4 Structure Can\_ControllerBaudrateConfigType

Configuration of CAN controller.

This structure is initialized by Tresos considering user settings. Used by `Can_ConfigType`. Passed to `Can_InitController()` at initialization.

## Declaration

## Software specification

```
typedef struct
{
    const uint32 u32ControlRegister,
    const uint32 u32ControlRegisterAlternate,
    const uint16 u16ControllerBaudRate,
    const Can_ControllerFdConfigType ControllerFD,
    const Can_ControllerCbtConfigType ControllerCbtRegister,
    const uint16 u16ControllerBaudRateConfigID,
} Can_ControllerBaudrateConfigType;
```

**Table 3-33. Structure Can\_ControllerBaudrateConfigType member description**

Member	Description
u32ControlRegister	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
u32ControlRegisterAlternate	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
u16ControllerBaudRate	Configured BaudRate in kbps.
ControllerFD	Content of the CANFD_CBT Register fields.
ControllerCbtRegister	Content of the CAN_CBT Register fields.
u16ControllerBaudRateConfigID	The ID of Controller baudrate configuration.

```
typedef struct
{
    const uint32 u32ControlRegister,
    const uint32 u32ControlRegisterAlternate,
} CanStatic_ControllerBaudrateConfigType;
```

**Table 3-34. Structure Can\_ControllerBaudrateConfigType member description**

Member	Description
u32ControlRegister	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
u32ControlRegisterAlternate	Content of the Control Register (CTRL) fields: PRES DIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.

### 3.8.4.5 Structure Can\_MBConfigContainerType

Type for storing Message Buffer configurations.

The MessageBufferConfigs array is sorted according to:

- HRHs first, HTHs next (AutoSAR requirement)
- Controller ID (HRHs and HTHs belonging to all controllers must be grouped together)
- Message ID (to ensure top priority IDs are first which means they will be serviced first)

## Declaration

```
typedef struct
{
    const Can_MBConfigObjectType *pMessageBufferConfigsPtr,
    const Can_HwHandleType uMessageBufferConfigCount
} Can_MBConfigContainerType;
```

**Table 3-35. Structure Can\_MBConfigContainerType member description**

Member	Description
pMessageBufferConfigsPtr	Pointer to the MB array .
uMessageBufferConfigCount	Number of elements in the array -( having 6 controllers with 64MBs each uint8 is not enough to store this value -> the type is extended to uint16).

### 3.8.4.6 Structure Can\_MBConfigObjectType

Type for storing information about Message Buffers (CAN hardware objs). Used by Can\_MBConfigContainerType.

#### Declaration

```
typedef struct
{
    Can_HwHandleType uIdMaskIndex,
    const uint8 u8ControllerId,
    CanIdType uIdType,
    Can_ObjType eMBType,
    Can_IdType uMessageId,
    const uint8 u8LocalPriority,
    uint32 u32HWObjID,
    uint8 u8FdPaddingValue,
    const uint8 u32CanMainFuncRWPPeriodRef,
    const uint16 u16MBOffsetAddr,
    const uint8 u8MBPayloadLength,
    const uint8 u8HWMBIndex,
    const boolean CanTriggerTransmitEnable
} Can_MBConfigObjectType;
```

**Table 3-36. Structure Can\_MBConfigObjectType member description**

Member	Description
uIdMaskIndex	Index into array of Can_FilterMaskType values (uint8/uint16), Current MB and the corresponding filter mask.
u8ControllerId	Controller ID (index into controller address array containing Can_ControllerPtrType).
uIdType	ID type: EXTENDED, STANDARD, MIXED.
eMBType	Receive/Transmit.
uMessageId	(extended identifier) (uint16/uint32). configurable by CanHardwareObject/ CanIdValue.
u8LocalPriority	Local priority bits used for arbitration.
u32HWObjID	HW Object ID.
u8FdPaddingValue	Padding value for MBs with the number of data bytes > 8. This parameter is used in FD mode

Table continues on the next page...

**Table 3-36. Structure Can\_MBConfigObjectType member description (continued)**

Member	Description
u32CanMainFuncRWPeriodRef	Read Write Period Reference .
u16MBOffsetAddr	Offset address of the MB in the message buffer memory area.
u8MBPayloadLength	Maximum data length (in bytes) of a CAN frame which is transmitted or received on the MB.
u8HWMBIndex	The index of the MB (also known as HOH) in the message buffer memory.
CanTriggerTransmitEnable	The parameter is used to detect the MB which run with trigger transmit feature.

### 3.8.4.7 Structure Can\_PduType

Type used to provide ID, DLC, SDU from CAN interface to CAN driver.  $HTH/HRH = ID + DLC + SDU$ .

#### Declaration

```
typedef struct
{
    Can_IdType id,
    PduIdType swPduHandle,
    uint8 length,
    uint8 *sdu
} Can_PduType;
```

**Table 3-37. Structure Can\_PduType member description**

Member	Description
id	CAN L-PDU = Data Link Layer Protocol Data Unit. Consists of Identifier, DLC and Data(SDU) It is uint32 for CAN_EXTENDEDID=STD_ON, else is uint16.
swPduHandle	The L-PDU Handle = defined and placed inside the CanIf module layer. Each handle represents an L-PDU, which is a constant structure with information for Tx/Rx processing.
length	DLC = Data Length Code (part of L-PDU that describes the SDU length).
sdu	CAN L-SDU = Link Layer Service Data Unit. Data that is transported inside the L-PDU.

### 3.8.4.8 Structure Can\_RxFifoTableIdConfigType

Rx Fifo Table IDs and Filter Masks.

#### Declaration

```
typedef struct
{
    const uint32 u32TableId,
```

```

    const uint32 u32TableFilterMask
} Can_RxFiFoTableIdConfigType;

```

**Table 3-38. Structure Can\_RxFiFoTableIdConfigType member description**

Member	Description
u32TableId	Table with the IDs specific for Rx Fifo.
u32TableFilterMask	Table with the Filter masks.

### 3.8.4.9 Structure Can\_ControllerFdConfigType

Can\_ControllerFdConfigType.

Type used to provide ID, DLC, SDU from CAN interface to CAN driver. HTH/HRH = ID+DLC+SDU.

#### Declaration

```

typedef struct
{
    uint32 u32CanFdEnable;
    uint32 u32CanFdBaudRate;
    uint32 u32CanFdCbtRegister;
    uint32 u32CanControllerTrcvDelayCompensation;
    uint32 u32CanControllerTxBitRateSwitch;
    uint32 u32CanFdCTRL2Register;
} Can_ControllerFdConfigType;

```

**Table 3-39. Structure Can\_ControllerFdConfigType member description**

Member	Description
u32CanFdEnable	This parameter is used to detect the FD feature enable or disable.
u32CanFdBaudRate	This parameter is used to store the baudrate of FD feature.
u32CanFdCbtRegister	This parameter is used to store the value which is written to CBT register.
u32CanControllerTrcvDelayCompensation	This parameter is used to store the value which is written in the FDCTRL register.(see more in the RM)
u32CanControllerTxBitRateSwitch	This parameter is used to detect the FD feature will run switch baudrate mode or not.
u32CanFdCTRL2Register	This parameter is used to configure some bit fields in CTRL2 register for the FD feature.

### 3.8.4.10 Structure Can\_ControllerCbtConfigType

Can\_ControllerCbtConfigType.

Type used to provide ID, DLC, SDU from CAN interface to CAN driver. HTH/HRH = ID+DLC+SDU.

Declaration

```
typedef struct
{
    uint32 u32CanCbtEnable;
    uint32 u32CanCbtBaudRate;
    uint32 u32CanCbtRegister;
} Can_ControllerCbtConfigType;
```

Table 3-40. Structure Can\_ControllerCbtConfigType member description

Member	Description
id	CAN L-PDU = Data Link Layer Protocol Data Unit. Consists of Identifier, DLC and Data(SDU) It is uint32 for CAN_EXTENDEDID=STD_ON, else is uint16.
swPduHandle	The L-PDU Handle = defined and placed inside the CanIf module layer. Each handle represents an L-PDU, which is a constant structure with information for Tx/Rx processing.
length	DLC = Data Length Code (part of L-PDU that describes the SDU length).
sdu	CAN L-SDU = Link Layer Service Data Unit. Data that is transported inside the L-PDU.

3.8.5 Types Reference

Types supported by the driver are as per AUTOSAR CAN Driver software specification Version 4.3 Rev0001 .

3.8.5.1 Typedef Can\_HwHandleType

Type:uint8

Represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects use extended range.

- used by "Can\_Write" function. The driver does not distinguish between Extended and Mixed transmission modes. Extended transmission mode of operation behaves the same as Mixed mode.

3.8.5.2 Typedef Can\_IdPtrType

Type: const uint32 \*const

Type for storing pointer to the Identifier Length Type.

- used by "Can\_ConfigType" structure (pointer to the FilterMasks).

### 3.8.5.3 Typedef Can\_IdType

**Type:** uint32

Type for storing the Identifier Length Type: Normal /Extended.

- used by "Can\_MessageBufferConfigObjectType" structure. The driver does not distinguish between Extended and Mixed transmission modes. Extended transmission mode of operation behaves the same as Mixed mode.

### 3.8.5.4 Typedef Can\_PCallbackType

Type for pointer to function.

**Type:** void(\*)

Type for pointer to function. Used for user handlers from plugin.

### 3.8.5.5 Typedef Can\_PtrControllerDescriptorType

**Type:** const Can\_ControllerDescriptorType \*

### 3.8.5.6 Typedef CanStatic\_PtrControllerDescriptorType

**Type:** const CanStatic\_ControllerDescriptorType \*

### 3.8.5.7 Typedef Can\_PtrMBConfigContainerType

**Type:** const Can\_MBConfigContainerType \*

## 3.9 Symbolic Names Disclaimer

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like:

```
#define <Container_Short_Name> <Container_ID>
```

For this reason it is forbidden to duplicate the name of such containers across the MCAL configuration, or to use names that may trigger other compile issues (e.g. match existing `#ifdefs` arguments).



## Chapter 4

# Tresos Configuration Plug-in

This chapter describes the Tresos configuration plug-in for the CAN Driver. The most of the parameters are described below.

### 4.1 Configuration elements of Can

Included forms:

- IMPLEMENTATION\_CONFIG\_VARIANT
- CanConfigSet
- CanGeneral
- CommonPublishedInformation

### 4.2 Form IMPLEMENTATION\_CONFIG\_VARIANT

Table 4-1. IMPLEMENTATION\_CONFIG\_VARIANT

<b>Description</b>	Defines whether pre-compile version is used. Using this option with VariantPostBuild value, Tresos can generate many CanConfigSet variants.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	VariantPreCompile, VariantPostBuild
<b>Default</b>	VariantPreCompile
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_PRECOMPILE_SUPPORT
<b>Autosar 4.3 Requirement</b>	NA

#### Note

This parameter permit to generate many configurations if VariantPostBuild is selected.

### 4.3 Form CanController

This configuration holds global control over the platform specific config control features. This container is implementation specific.

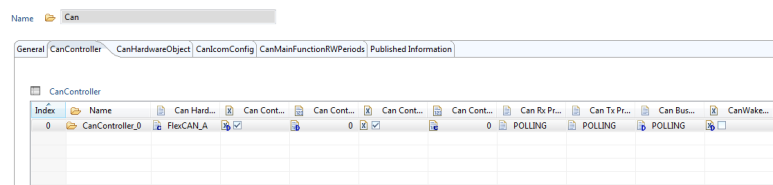


Figure 4-1. TRESOS Plugin snapshot for CanConfigSet form.

#### 4.3.1 Can-Controller Parameters

**Can-Controller** parameters, their possible values and meaning are described in the following text. The Can-Controller parameters are implemented as constant structures and arrays stored in flash memory of the MCU.

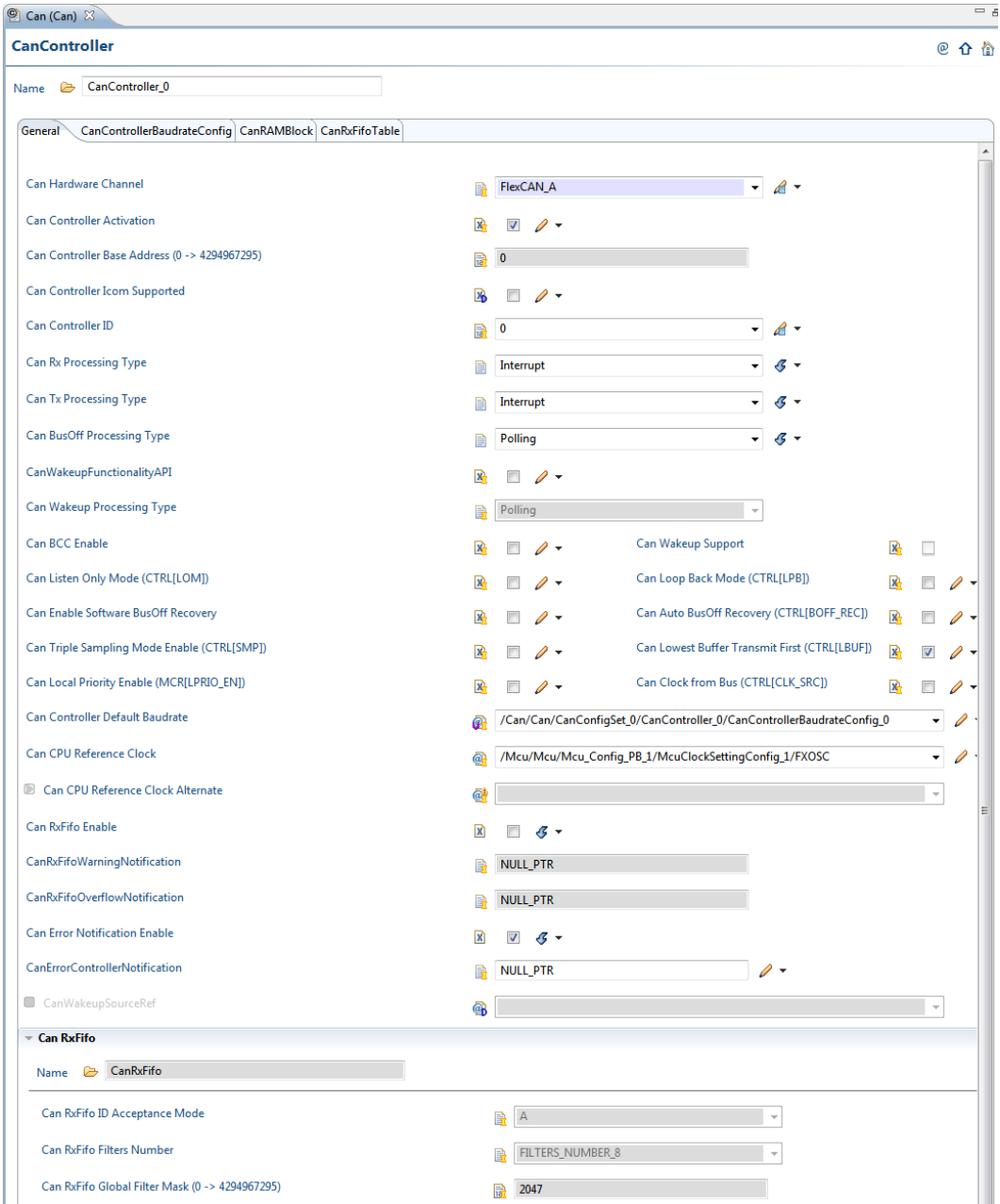


Figure 4-2. Can Controller Parameters

4.3.1.1 CanHwChannel

Table 4-2. CanHwChannel

Description	Specifies which one of the on-chip FlexCan interface is associated with the controller ID.
Class	Implementation Specific Parameter
Range	FlexCAN_A, FlexCAN_B, ...
Default	FlexCAN_A

Table continues on the next page...

**Table 4-2. CanHwChannel (continued)**

<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = { ..... /* Can Controller Offset on chip */ FLEXCAN_A_OFFSET, ..... }  CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { ..... /* Can Controller Offset on chip */ FLEXCAN_A_OFFSET, ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter set the Can hardware channel for which the settings are implemented. The order from hardware (Can\_A first and Can\_B second) is not mandatory to be respected in the CanController list from Tresos plugin.

**4.3.1.2 CanControllerActivation****Table 4-3. CanControllerActivation**

<b>Description</b>	Defines if a CAN controller is used in the configuration.
<b>Class</b>	Autosar Parameter
<b>Range</b>	True , False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00315

**Note**

This parameter permit to use the current settings for one of Can controllers. If this control is set to ‘false’ no Can controller initialization is made (NULL\_PTR).

### 4.3.1.3 CanControllerBaseAddress

**Table 4-4. CanControllerBaseAddress**

<b>Description</b>	Specifies the CAN controller base address.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 .. 4294967295
<b>Default</b>	0
<b>Source File</b>	NA
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00382

#### Note

This requirement is not supported by the Can Driver.

### 4.3.1.4 CanControllerId

**Table 4-5. CanControllerId**

<b>Description</b>	This parameter provides the controller ID which is unique in a given CAN Driver. The value for this parameter starts with 0 and continues without any gaps.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 .. 255
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CanController_0 0U /* Default configuration for FlexCAN_A */
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00316

#### Note

This parameter set an ID for the current Can controller configuration. Also a define is generated for each Can controller.

### 4.3.1.5 CanRxProcessing

**Table 4-6. CanRxProcessing**

<b>Description</b>	Specifies if RX events are polled inside Can_MainFunction_Read or cause an interrupt.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Polling , Interrupt
<b>Default</b>	Polling

*Table continues on the next page...*

**Table 4-6. CanRxProcessing (continued)**

<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT][index] = { ..... /* ===== Controller Options ===== */ /* RxPoll Enabled */ CAN_CONTROLLERCONFIG_RXPOL_EN_U32 ..... }  CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT][index] = { ..... /* ===== Controller Options ===== */ /* RxPoll Enabled */ CAN_CONTROLLERCONFIG_RXPOL_EN_U32 ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00317

**Note**

This parameter set how it is implemented the handling of Rx confirmation events: by polling or by interrupt.

**4.3.1.6 CanTxProcessing****Table 4-7. CanTxProcessing**

<b>Description</b>	Specifies if TX events are polled inside Can_MainFunction_Write or cause an interrupt.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Polling , Interrupt
<b>Default</b>	Polling
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = { ..... /* ===== Controller Options ===== */ /* TxPoll Enabled */ CAN_CONTROLLERCONFIG_TXPOL_EN_U32   ..... } </pre>

*Table continues on the next page...*

**Table 4-7. CanTxProcessing (continued)**

	<pre> CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { ..... /* ===== Controller Options ===== */ /* TxPoll Enabled */ CAN_CONTROLLERCONFIG_TXPOL_EN_U32   ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00318

**Note**

This parameter set how it is implemented the handling of Tx confirmation events: by polling or by interrupt.

**4.3.1.7 CanBusOffProcessing****Table 4-8. CanBusOffProcessing**

<b>Description</b>	Specifies if bus-off events are polled inside Can_Main_Function_BusOff or cause an interrupt.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Polling , Interrupt
<b>Default</b>	Polling
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_[index]] = { ..... /* ===== Controller Options ===== */ /* BusOffPoll Enabled */ CAN_CONTROLLERCONFIG_BOPOL_EN_U32   ..... }  CONST(CanStatic_ControllerDescriptorType, CAN_CONST) StaticControllerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { ..... /* ===== Controller Options ===== */ /* BusOffPoll Enabled */ CAN_CONTROLLERCONFIG_BOPOL_EN_U32   ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00314

**Note**

This parameter set how it is implemented the handling of BusOff confirmation events: by polling or by interrupt.

**4.3.1.8 CanListenOnlyMode****Table 4-9. CanListenOnlyMode**

<b>Description</b>	Enables the Listen only mode.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet_PC[CAN_MAXCONTROLLERCOUNT] = { /* ===== Control Register - CTRL ===== */ ..... /* CTRL[LOM] - Listen only mode */ ..... }  CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { /* ===== Control Register - CTRL ===== */ ..... /* CTRL[LOM] - Listen only mode */ ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode.

**4.3.1.9 CanLoopBackMode****Table 4-10. CanLoopBackMode**

<b>Description</b>	Enables the Loop Back Mode.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False

*Table continues on the next page...*



**Table 4-10. CanLoopBackMode (continued)**

<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet_PC_[CAN_MAXCONTROLLERCOUNT] = {  /* ===== Control Register - CTRL ===== */  .....  /* CTRL[LPB] - Loop-back mode */  .....  }  CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = {  /* ===== Control Register - CTRL ===== */  .....  /* CTRL[LPB] - Loop-back mode */  .....  } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic "1").

**4.3.1.10 CanSoftwareBusOffRecovery****Table 4-11. CanSoftwareBusOffRecovery**

<b>Description</b>	Enables Automatic Bus Recovery Off.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(Can_ControllerDescriptorType, CAN_CONST) ControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {  /* ===== Controller Options ===== */  /* Software BusOff Recovery */  CAN_CONTROLLERCONFIG_BUSOFFSWREC_U32    .....  } </pre>

*Table continues on the next page...*

**Table 4-11. CanSoftwareBusOffRecovery  
(continued)**

	<pre> CONST(Can_ControllerDescriptorType, CAN_CONST) ControllerDescriptors[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { /* ===== Controller Options ===== */ /* Software BusOff Recovery */ CAN_CONTROLLERCONFIG_BUSOFFSWREC_U32 I ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Enables Software Bus Off recovery when automatic recovering from BusOff state is disabled for CAN controller.

**4.3.1.11 CanAutoBusOffRecovery****Table 4-12. CanAutoBusOffRecovery**

<b>Description</b>	Enables Automatic Bus Recovery Off.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet_PC_[CAN_MAXCONTROLLERCOUNT] = { /* ===== Control Register - CTRL ===== */ ..... /* CTRL[BOFF_REC] - Bus off recovery */ ..... }  CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { /* ===== Control Register - CTRL ===== */ ..... /* CTRL[BOFF_REC] - Bus off recovery */ ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Enable/Disable automatic BusOff recovery (CTRL[BOFF\_REC] bit). 0(Checked) = Automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. 1(Unchecked) = Automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated(zero) by the user.

**4.3.1.12 CanTripleSamplingEnable****Table 4-13. CanTripleSamplingEnable**

<b>Description</b>	Enables acquisition of 3 samples and majority voting for the value of received bit.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet_PC[CAN_MAXCONTROLLERCOUNT] = { /* ===== Control Register - CTRL ===== */ ..... /* CTRL[SMP] - Sampling mode */ ..... }  CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { /* ===== Control Register - CTRL ===== */ ..... /* CTRL[SMP] - Sampling mode */ ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This bit defines the sampling mode of CAN bits at the Rx input.  
1 - Enables acquisition of 3 samples and majority voting for the value of received bit. 0 - Just one sample is used to determine the bit value.

### 4.3.1.13 CanLowestBuffTransmitFirst

**Table 4-14. CanLowestBuffTransmitFirst**

<b>Description</b>	This parameter defines the ordering mechanism for MB transmission.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet_PC[CAN_MAXCONTROLLERCOUNT] = { /* ===== Control Register - CTRL ===== */ .....  /* CTRL[LBUF] - Lowest Buffer Transmitted First */ .....  }  CONST(CanStatic_ControllerBaudrateConfigType, CAN_CONST) StaticControllerBaudrateCfgSet[index]_PB[CAN_MAXCONTROLLERCOUNT_[index]] = { /* ===== Control Register - CTRL ===== */ .....  /* CTRL[LBUF] - Lowest Buffer Transmitted First */ .....  } </pre>
<b>Autosar 4.3 Requirement</b>	NA

#### Note

CTRL[LBUF]. This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the MCR[LPRIO\_EN] bit doesn't affect the priority arbitration.

### 4.3.1.14 CanLocalPriorityEn

**Table 4-15. CanLocalPriorityEn**

<b>Description</b>	This field is used when MCR[LPRIO_EN] is set and makes sense only for Tx MBs.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre> CONST(Can_ControllerDescriptorType, CAN_CONST) ControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_0] = { /* ===== Controller Options ===== */ ..... </pre>

*Table continues on the next page...*

**Table 4-15. CanLocalPriorityEn  
(continued)**

	<pre>/* Local Priority Feature */ CAN_CONTROLLERCONFIG_LPRIO_EN_U32 { }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

MCR[LPRIO\_EN]. This bit controls whether the local priority feature is enabled or not.

**4.3.1.15 CanClockFromBus****Table 4-16. CanClockFromBus**

<b>Description</b>	Switches the source clock for the module to the system bus (rather than crystal).
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre>CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {  /* ===== Control Register - CTRL ===== */  .....  /* CTRL[CLK_SRC] - Clock source */  .....  }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Switches the source clock for the module to the system bus (rather than crystal). 1 = The CAN engine clock source is the bus clock.(from MCU). 0 = The CAN engine clock source is the oscillator clock.

**Note**

The Can module of Rainier has a issue about source clock when use the oscillator clock. It only uses the bus clock.

### 4.3.1.16 CanCpuClockRef

**Table 4-17. CanCpuClockRef**

<b>Description</b>	Reference to the CPU clock configuration, which is set in the MCU driver configuration.
<b>Class</b>	Autosar Parameter
<b>Range</b>	NA
<b>Default</b>	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig/McuClockReferencePoin_0
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00313

#### Note

Reference to the CPU clock configuration, which is set in the MCU driver configuration. MCU plugin need to be added and then give the reference to it.

#### Note

"CanCpuClockRef" it is extracted from MCU and it is used exclusively for the computation of baudrate configuration parameters.

### 4.3.1.17 CanControllerRXFifoEnable

**Table 4-18. CanControllerRXFifoEnable**

<b>Description</b>	Defines if RX FIFO feature of CAN controller is used in the configuration.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_A_FIFO_EN STD_ON / STD_OFF
<b>Autosar 4.3 Requirement</b>	NA

#### Note

Defines if RX FIFO feature of CAN controller is used in the configuration. If FIFO feature of CAN controller is enabled, First 8 Message Buffers will be used by FIFO engine.

### 4.3.1.18 CanRxFifoWarningNotification

**Table 4-19. CanRxFifoWarningNotification**

<b>Description</b>	Defines the handler for Rx Fifo warning.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	String
<b>Default</b>	NULL_PTR
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	extern FUNC(void, CAN_CODE) handler_name (void);
<b>Autosar 4.3 Requirement</b>	NA

### 4.3.1.19 CanRxFifoOverflowNotification

**Table 4-20. CanRxFifoOverflowNotification**

<b>Description</b>	Defines the handler for Rx Fifo overflow.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	String
<b>Default</b>	NULL_PTR
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	extern FUNC(void, CAN_CODE) handler_name (void);
<b>Autosar 4.3 Requirement</b>	NA

### 4.3.1.20 CanErrorControllerNotification

**Table 4-21. CanErrorControllerNotification**

<b>Description</b>	Defines the handler for error controller.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	String
<b>Default</b>	NULL_PTR
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	extern FUNC(void, CAN_CODE) handler_name (void);
<b>Autosar 4.3 Requirement</b>	NA

### 4.3.1.21 CanCpuClockRef\_Alternate

**Table 4-22. CanCpuClockRef\_Alternate**

<b>Description</b>	Reference to the CPU clock configuration, which is set in the MCU driver configuration. This parameter is enabled only if "CanClockFromBus" is set to true.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	NA
<b>Default</b>	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig/McuClockReferencePoin_0
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

#### Note

Reference to the CPU clock configuration, which is set in the MCU driver configuration. This parameter is available/editable only if "Can.CanConfig.DualClockMode" is set to STD\_ON from Resource files, "CanEnableDualClockMode" is set to true and "CanClockFromBus" = "true". MCU plugin need to be added and then give the reference to it.

#### Note

"CanCpuClockRef\_Alternate" it is extracted from MCU and it is used exclusively for the computation of baudrate configuration parameters. It is user responsibility to synchronize the value selected in this field with "CanClockFromBus" value

### 4.3.1.22 CanBccSupport

**Table 4-23. CanBccSupport**

<b>Description</b>	Defines if Backwards Compatibility Configuration (BCC) feature of CAN controller is used in the configuration.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_BCC_SUPPORT_ENABLE STD_ON / STD_OFF
<b>Autosar 4.3 Requirement</b>	NA



## Note

If BCC feature of CAN controller is enabled, Individual Rx masking and queue feature are disabled. Else, Individual Rx masking and queue feature are enabled. When Backwards Compatibility Configuration (BCC) feature of CAN controller is used in the configuration, below should be the configuration of CanHardwareObject.

CanFilterMask configuration in CanController container:

```
=====
```

```
=====
```

CanFilterMask\_0

CanFilterMask\_1

CanFilterMask\_2

CanHardwareObject\_0 to CanHardwareObject\_13 and  
CanHardwareObject\_16 to CanHardwareObject\_32/  
CanHardwareObject\_63:

```
=====
```

```
=====
```

```
=====
```

```
=====
```

CanFilterMask\_0 should be selected in CanFilterMaskRef

CanHardwareObject\_14 :

```
=====
```

CanFilterMask\_1 should be selected in CanFilterMaskRef

CanHardwareObject\_15 :

```
=====
```

CanFilterMask\_2 should be selected in CanFilterMaskRef

\*/

### 4.3.1.23 CanErrorControllerNotifEn

**Table 4-24. CanErrorControllerNotifEn**

<b>Description</b>	Enables/Disables the Error Controller Notification. If Disabled, no error interrupt or notification shall take place.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_ERROR_NOTIFICATION_ENABLE STD_ON / STD_OFF
<b>Autosar 4.3 Requirement</b>	NA

#### Note

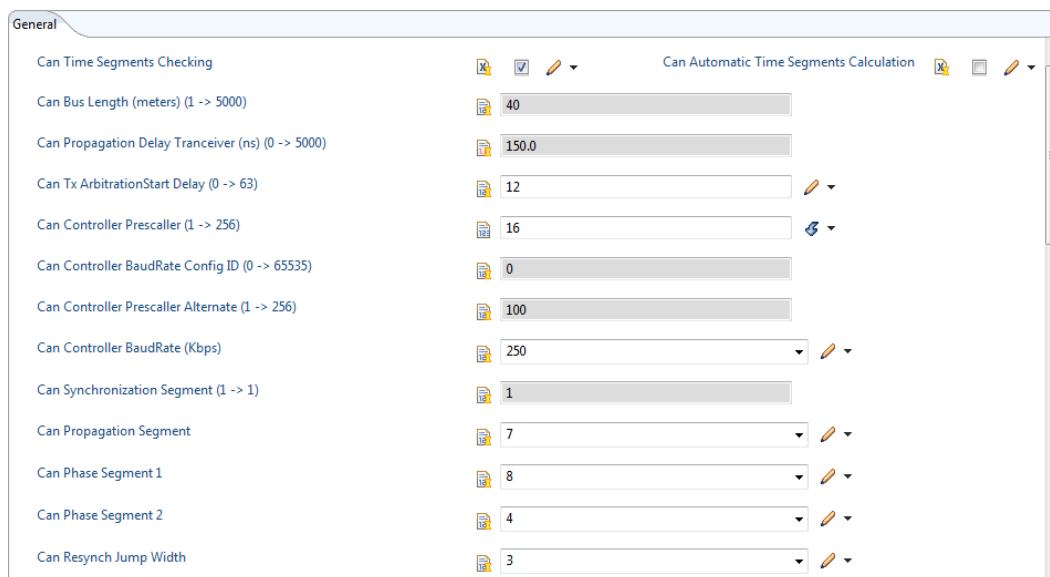
Not AutoSar Required.

### 4.3.1.24 CanControllerDefaultBaudrate

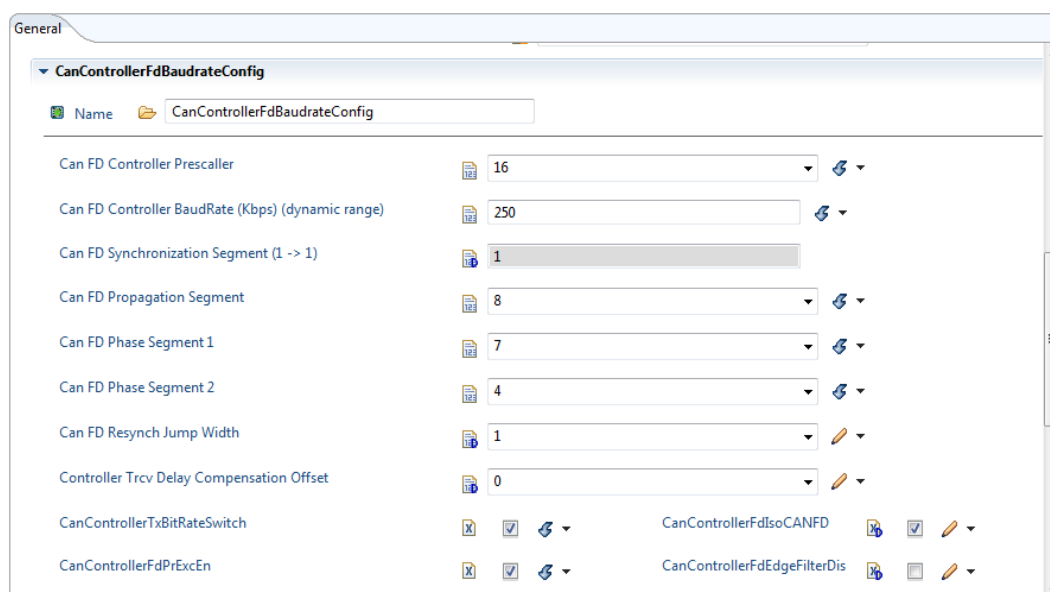
**Table 4-25. CanControllerDefaultBaudrate**

<b>Description</b>	Reference to baudrate configuration container configured for the Can Controller.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	NA
<b>Default</b>	/Can/Can/CanConfigSet_0/CanController_0/CanControllerBaudrateConfig_0
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00435

### 4.3.1.25 CanControllerBaudrateConfig



**Figure 4-3. CanControllerBaudrateConfig**



**Figure 4-4. CanControllerFDBaudrateConfig**

#### Note

Can-Controller Baudrate Config parameters, their possible values and meaning are described in the following text.

The Can-Controller BaudRate parameters are implemented as constant structures and arrays stored in flash memory of the MCU.

**4.3.1.25.1 CanControllerCheckCanStandard****Table 4-26. CanControllerCheckCanStandard**

<b>Description</b>	If enabled, Can Plugin checks that CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanSyncJumpWidth settings match the CAN Standard Compliant Bit Time Segment Settings.
<b>Class</b>	Implementation Specific Parameter.
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	NA
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter is used in the Can\_Cfg.c and Can\_PBcfg.c files validating time segments values if it is enabled.

**4.3.1.25.2 CanAdvancedSetting****Table 4-27. CanAdvancedSetting**

<b>Description</b>	If True initiates the derivation of the Can bit timing values from the CanControllerBaudRate parameter and source clock value.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

**Note**

If this parameter is set to “True” then “CanControllerPropSeg”, “CanControllerSeg1”, “CanControllerSeg2” and “CanSyncJumpWidth” are disabled and these values are calculated indirectly. In the same time two another parameters are enabled: "BusLength" and "PropagationDelayOfTranceiver".

**4.3.1.25.3 CanBusLength****Table 4-28. CanBusLength**

<b>Description</b>	Specifies the Can bus length in meters. This parameter is used for PropSeg calculation when “CanAdvancedSetting” is set to true.
--------------------	----------------------------------------------------------------------------------------------------------------------------------

*Table continues on the next page...*

**Table 4-28. CanBusLength (continued)**

<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 – 5000
<b>Default</b>	40
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter is available only if “AdvancedSetting” is set to true.

**4.3.1.25.4 CanPropDelayTranceiver****Table 4-29. PropagationDelayOfTranceiver**

<b>Description</b>	Specifies the propagation delay in ns for the Can transceiver. This parameter is used for PropSeg calculation when “CanAdvancedSetting” is set to true.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 – 5000
<b>Default</b>	150
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter is available only if “CanAdvancedSetting” is set to true.

**4.3.1.25.5 CanTxArbitrationStartDelay****Table 4-30. CanTxArbitrationStartDelay**

<b>Description</b>	This parameter indicates how many CAN bits the Tx arbitration process start point can be delayed from the first bit of CRC field on CAN bus.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 63
<b>Default</b>	12
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

### 4.3.1.25.6 CanControllerPrescaler

**Table 4-31. CanControllerPrescaler**

<b>Description</b>	Specifies the prescaler for the controller. The calculation of the resulting CanControllerTimeQuanta value depending on module clocking and prescaler shall be done offline. $\text{Prescaler} = \text{FreqCanClk} / \text{FreqTq}$ ; $\text{FreqTq} = 1 / \text{CanControllerTimeQuanta}$ .
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 - 256
<b>Default</b>	100
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

### 4.3.1.25.7 CanControllerPrescaler\_Alt

**Table 4-32. CanControllerPrescaler\_Alt**

<b>Description</b>	Specifies the alternate prescaler for the controller. The calculation of the resulting CanControllerTimeQuanta_Alternate value depending on module clocking and prescaler shall be done offline. $\text{Prescaler} = \text{FreqCanClk} / \text{FreqTq}$ ; $\text{FreqTq} = 1 / \text{CanControllerTimeQuanta}$ .
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 - 256
<b>Default</b>	100
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

#### Note

This parameter is available/editable only if "Can.CanConfig.DualClockMode" is set to STD\_ON from Resource files and "CanEnableDualClockMode" is set to true.

### 4.3.1.25.8 CanControllerBaudRate

**Table 4-33. CanControllerBaudRate**

<b>Description</b>	Specifies the baudrate of the controller in kbps.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 1000
<b>Default</b>	20
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00005

#### 4.3.1.25.9 CanControllerPropSeg

**Table 4-34. CanControllerPropSeg**

<b>Description</b>	Propogation delay in time quanta.
<b>Class</b>	Autosar Parameter
<b>Range</b>	1 - 8
<b>Default</b>	5
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = { ..... /* ===== Control Register - CTRL ===== */ ..... /* CTRL[PROPSEG] - Propagation segment */ 5U , ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00073

#### Note

It is used to compensate the physical delay within the CAN network (CTRL[PROPSEG] - 1..8).

#### 4.3.1.25.10 CanControllerSeg1

**Table 4-35. CanControllerSeg1**

<b>Description</b>	Specifies Phase Segment 1
<b>Class</b>	Autosar Parameter
<b>Range</b>	1 - 8
<b>Default</b>	5
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = { ..... /* ===== Control Register - CTRL ===== */ ..... /* CTRL[PSEG1] - Segment 1 */ 5U , ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00074

**Note**

Specifies the Phase Segment 1 in time quantas (CTRL[PSEG1] = 1..8). PHASE\_BUF\_SEG1 = PSEG1 \* Tq . The PHASE\_BUF\_SEG1 valid values are 1-8 Tq.

**4.3.1.25.11 CanControllerSeg2****Table 4-36. CanControllerSeg2**

<b>Description</b>	Specifies Phase Segment 2
<b>Class</b>	Autosar Parameter
<b>Range</b>	2 - 8
<b>Default</b>	6
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = { ..... /* ===== Control Register - CTRL ===== */ ..... /* CTRL[PSEG2] - Segment 2 */ 6U , ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00075

**Note**

Specifies Phase Segment 2 in time quantum (CTRL[PSEG2] = 2..8). PHASE\_BUF\_SEG2 = PSEG2 \* Tq . The PHASE\_BUF\_SEG2 valid values are 2-8 Tq.

**4.3.1.25.12 CanControllerSyncJumpWidth****Table 4-37. CanControllerSyncJumpWidth**

<b>Description</b>	Specifies Synchronization Jump Width.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 - 4
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC[CAN_MAXCONTROLLERCOUNT] = { ..... /* ===== Control Register - CTRL ===== */ </pre>

*Table continues on the next page...*



**Table 4-37. CanControllerSyncJumpWidth  
(continued)**

	<pre> ..... /* CTRL[RJW] - Resynchronization Jump Width */ ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00383

**Note**

Specifies Synchronization Jump Width: CTRL[RJW] = 1..4.

**4.3.1.25.13 CanControllerFdBaudrateConfig****Table 4-38. CanControllerFdBaudrateConfig**

<b>Description</b>	Enable or disable config for CAN_FD and also FD mode
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {     #if (CAN_FD_MODE_ENABLE == STD_ON){         /*true;*/         .....         .....     } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies Synchronization Jump Width: CTRL[RJW] = 0..3.

**4.3.1.25.14 CanControllerFdBaudRate****Table 4-39. CanControllerFdBaudRate**

<b>Description</b>	Specifies the data segment baud rate of the controller in kbps.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 8000
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c

*Table continues on the next page...*

**Table 4-39. CanControllerFdBaudRate (continued)**

<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)0U, /* 50kbps baud rate */ ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**4.3.1.25.15 CanControllerPropSeg****Table 4-40. CanControllerPropSeg**

<b>Description</b>	Specifies propagation delay in time quantas.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... .....(uint32)(0U &lt;&lt; (IPV_FlexCAN_FD_PROPSEG_OFFSET)) ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

It is used to compensate the physical delay within the CAN network (CTRL[PROPSEG] - 1..8).

**4.3.1.25.16 CanControllerFDPrescaller****Table 4-41. CanControllerFDPrescaller**

<b>Description</b>	Specifies the prescaler for the controller in FD mode. The calculation of the resulting CanControllerTimeQuanta value depending on module clocking and prescaler shall be done offline. $\text{Prescaler} = \text{FreqCanClk} / \text{FreqTq}$ ; $\text{FreqTq} = 1 / \text{CanControllerTimeQuanta}$ .
<b>Class</b>	Implementation Specific Parameter

*Table continues on the next page...*

**Table 4-41. CanControllerFDPrescaler (continued)**

<b>Range</b>	1 - 256
<b>Default</b>	100
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)(0U &lt;&lt; IPV_FlexCAN_FD_PROPSEG_OFFSET)   ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

#### 4.3.1.25.17 CanControllerSeg1

**Table 4-42. CanControllerSeg1**

<b>Description</b>	Specifies phase segment 1 in time quantas.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)(0U &lt;&lt; IPV_FlexCAN_FD_PSEG1_OFFSET)   ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

#### Note

Specifies the Phase Segment 1 in time quantas (CTRL[PSEG1] = 1..8).  $PHASE\_BUF\_SEG1 = PSEG1 * Tq$ . The PHASE\_BUF\_SEG1 valid values are 1-8 Tq.

**4.3.1.25.18 CanControllerSeg2****Table 4-43. CanControllerSeg2**

<b>Description</b>	Specifies phase segment 2 in time quantas.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)(0U&lt;&lt;IPV_FlexCAN_FD_PSEG2_OFFSET)   ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies Phase Segment 2 in time quantum (CTRL[PSEG2] = 2..8). PHASE\_BUF\_SEG2 = PSEG2 \* Tq . The PHASE\_BUF\_SEG2 valid values are 2-8 Tq.

**4.3.1.25.19 CanControllerSyncJumpWidth****Table 4-44. CanControllerSyncJumpWidth**

<b>Description</b>	Specifies the synchronization jump width for the controller in time quantas.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)(0U&lt;&lt;IPV_FlexCAN_FD_SJW_OFFSET) , /*Sync jump width*/ ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies Synchronization Jump Width: CTRL[RJW] = 1..4.

**4.3.1.25.20 CanControllerTrcvDelayCompensationOffset****Table 4-45. CanControllerTrcvDelayCompensationOffset**

<b>Description</b>	Specifies the Transceiver Delay Compensation Offset in ns. If not specified Transceiver Delay Compensation is disabled.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 400
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)0U, /*TRCV DELAY*/ ..... }</pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00480

**Note**

Specifies Synchronization Jump Width: CTRL[RJW] = 0..3.

0x0–0x1F — Offset value defining the distance between the measured delay from m\_can\_tx to m\_can\_rx and the secondary sample point. Valid values are 0 to 31 M\_CAN clock periods.

**4.3.1.25.21 CanControllerTxBitRateSwitch****Table 4-46. CanControllerTxBitRateSwitch**

<b>Description</b>	CanControllerTxBitRateSwitch it is used to enable a feature, which can switch baudrate transmtion of data phase (which has a different value comparing with the nominal baudrate). Specifies if the bit rate switching shall be used for transmissions. If FALSE: CAN FD frames shall be sent without bit rate switching.
<b>Class</b>	Implementation Specific Parameter.
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	NA
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = {</pre>

*Table continues on the next page...*

**Table 4-46. CanControllerTxBitRateSwitch  
(continued)**

	<pre> ..... #if (CAN_FD_MODE_ENABLE == STD_ON){ ..... (uint32)(0U&lt;&lt;IPV_FlexCAN_FD_BRS_OFFSET) /*false ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter is used in the Can\_Cfg.c and Can\_PBcfg.c files validating time segments values if it is enabled.

**4.3.1.25.22 CanControllerCbtEnable****Table 4-47. CanControllerCbtEnable**

<b>Description</b>	CanControllerTxBitRateSwitch it is used to enable FD data baudrate (which has a different value comparing with the nominal baudrate).
<b>Class</b>	Implementation Specific Parameter.
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	NA
<b>Source Representation</b>	<pre> static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)(1U &lt;&lt;FLEXCAN_CBT_OFFSET), ..... } </pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter is used in the Can\_Cfg.c and Can\_PBcfg.c files validating time segments values if it is enabled. When FD is enabled the baudrate is calculated from CBT fields. The data baudrate it is calculated from FDCBT. IF BRS is checked then the message will be sent with two baudrates: nominal baudrate from cbt and data baudrate from FDCBT.

## 4.3.1.25.23 CanControllerBaudRate

Table 4-48. CanControllerBaudRate

<b>Description</b>	Specifies the buadrate of the controller in kbps.
<b>Class</b>	Implementation Specific Parameter.
<b>Range</b>	0 - 16000
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)0U, /* 50kbps baud rate */ ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

## 4.3.1.25.24 CanControllerCbtPropSeg

Table 4-49. CanControllerCbtPropSeg

<b>Description</b>	Specifies propagation delay in time quantas.
<b>Class</b>	Autosar Parameter
<b>Range</b>	1 - 64
<b>Default</b>	5
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)(0U &lt;&lt; FLEXCAN_CBT_PROPSEG_OFFSET)   ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

It is used to compensate the physical delay within the CAN network (CTRL[PROPSEG] - 1..8).

**4.3.1.25.25 CanControllerCbtPrescaler****Table 4-50. CanControllerCbtPrescaler**

<b>Description</b>	The calculation of the resulting CanControllerTimeQuanta value depending on module clocking and prescaler shall be done offline. $\text{Prescaler} = \text{FreqCanClk} / \text{FreqTq}$ ; $\text{FreqTq} = 1 / \text{CanControllerTimeQuanta}$ .
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 - 1023
<b>Default</b>	100
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)(99U &lt;&lt; FLEXCAN_FD_PRESDIV_CBT_OFFSET).I, ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**4.3.1.25.26 CanControllerCbtSeg1****Table 4-51. CanControllerCbtSeg1**

<b>Description</b>	Specifies phase segment 1 in time quantas.
<b>Class</b>	Autosar Parameter
<b>Range</b>	1 - 32
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)(0U &lt;&lt; FLEXCAN_CBT_PSEG1_OFFSET) I ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies the Phase Segment 1 in time quantas ( $\text{CTRL}[\text{PSEG1}] = 1..8$ ).  $\text{PHASE\_BUF\_SEG1} = \text{PSEG1} * \text{Tq}$ . The  $\text{PHASE\_BUF\_SEG1}$  valid values are 1-8  $\text{Tq}$ .



### 4.3.1.25.27 CanControllerCbtSeg2

**Table 4-52. CanControllerCbtSeg2**

<b>Description</b>	Specifies phase segment 2 in time quantas.
<b>Class</b>	Autosar Parameter
<b>Range</b>	1 - 32
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)(0U&lt;&lt;FLEXCAN_CBT_PSEG2_OFFSET) /*Time segment 2 */ ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

#### Note

Specifies Phase Segment 2 in time quantum ( $CTRL[PSEG2] = 2..8$ ).  $PHASE\_BUF\_SEG2 = PSEG2 * Tq$ . The  $PHASE\_BUF\_SEG2$  valid values are 2-8  $Tq$ .

### 4.3.1.25.28 CanControllerSyncJumpWidthCbt

**Table 4-53. CanControllerSyncJumpWidthCbt**

<b>Description</b>	Specifies the synchronization jump width for the controller in time quantas.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	1 - 16
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CBT_ENABLE == STD_ON){ ..... (uint32)(0U&lt;&lt;FLEXCAN_CBT_SJW_OFFSET) /*Sync jump width*/ ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies Synchronization Jump Width: CTRL[RJW] = 1..4.

**4.3.1.25.29 CanControllerFdIsoCANFD****Table 4-54. CanControllerFdIsoCANFD**

<b>Description</b>	The CanControllerFdIsoCANFD is used to selecting ISO or none-ISO for CAN FD of IPV_FLEXCAN. In the ASR421, it is a none ASR parameter, but it is put as a none variant parameter.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre>static CONST(Can_ControllerBaudrateConfigType, CAN_CONST) ControllerBaudrateConfigs_PC_0[CAN_CFGSET0_MAX_BAUDRATE_FC_A_0] = { ..... #if (CAN_CONTROLLER_FD_ISO_CANFD == STD_ON) ..... (uint32)(1U&lt;&lt;FLEXCAN_FD_CTRL2_STFCNTEN_SHIFT_U32) ..... }</pre>
<b>Autosar 4.3 Requirement</b>	NA

**4.3.1.25.30 CanControllerFdPrExcEn****Table 4-55. CanControllerFdPrExcEn**

<b>Description</b>	This bit enables the Protocol Exception feature for the CAN FD mode.
<b>Class</b>	Implementation Specific Parameter.
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	NA
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

**4.3.1.25.31 CanControllerFdEdgeFilterDis****Table 4-56. CanControllerFdEdgeFilterDis**

<b>Description</b>	This parameter configures the Edge Filter used during the bus integration state.
<b>Class</b>	Implementation Specific Parameter.
<b>Range</b>	True, False
<b>Default</b>	False

Table continues on the next page...

**Table 4-56. CanControllerFdEdgeFilterDis (continued)**

Source File	NA
Source Representation	NA
Autosar 4.3 Requirement	NA

### 4.3.1.26 Can RxFifo

**Can RxFifo**

Name CanRxFifo

---

Can RxFifo ID Acceptance Mode A

Can RxFifo Filters Number FILTERS\_NUMBER\_8

Can RxFifo Global Filter Mask (0 -> 4294967295) 2047

**Figure 4-5. Can RxFifo Filters Number and Can RxFifo Global Filter Mask**

**CanController**




Name CanController\_0


General | CanControllerBaudrateConfig | CanFilterMask | **CanRxFifoTable**

CanRxFifoTable













Index	Name	Can RxFif...	Can RxFif...	Can RxFif...	Can RxFif...	Can Rx...	Can RxFif...
0	CanRxFifoTable_0	255	255	255	255	Standard	2047
1	CanRxFifoTable_1	255	255	255	255	Standard	2047
2	CanRxFifoTable_2	255	255	255	255	Standard	2047
3	CanRxFifoTable_3	255	255	255	255	Standard	2047
4	CanRxFifoTable_4	255	255	255	255	Standard	2047
5	CanRxFifoTable_5	255	255	255	255	Standard	2047
6	CanRxFifoTable_6	255	255	255	255	Standard	2047
7	CanRxFifoTable_7	17	255	255	255	Standard	2047

**Figure 4-6. Can RxFifo**




**CanRxFifoTable**   


Name  CanRxFifoTable\_0

**General**



















Can RxFifo Identifier 0		255	
Can RxFifo Identifier 1		255	
Can RxFifo Identifier 2		255	
Can RxFifo Identifier 3		255	
Can RxFifo Frame Accepted Mode		Standard	
Can RxFifo Filter Mask		2047	

**Figure 4-7. Can CanRxFifoTable - CanControllerRxFifoEnable = FALSE**




**CanRxFifoTable**   


Name  CanRxFifoTable\_0

**General**

















Can RxFifo Identifier 0		255			
Can RxFifo Identifier 1		255			
Can RxFifo Identifier 2		255			
Can RxFifo Identifier 3		255			
Can RxFifo Frame Accepted Mode		Standard			
Can RxFifo Filter Mask		2047			

**Figure 4-8. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = A**




**CanRxFifoTable**   


Name  CanRxFifoTable\_0

**General**


















Can RxFifo Identifier 0	 255	 
Can RxFifo Identifier 1	 255	 
Can RxFifo Identifier 2	 255	
Can RxFifo Identifier 3	 255	
Can RxFifo Frame Accepted Mode	 Standard	 
Can RxFifo Filter Mask	 2047	 

**Figure 4-9. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = B**

**CanRxFifoTable**   

Name  CanRxFifoTable\_0

**General**

Can RxFifo Identifier 0	 255	 
Can RxFifo Identifier 1	 255	 
Can RxFifo Identifier 2	 255	 
Can RxFifo Identifier 3	 255	 
Can RxFifo Frame Accepted Mode	 Standard	
Can RxFifo Filter Mask	 2047	 

**Figure 4-10. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = C**

CanRxFifoTable

@↑🏠

Name📁

CanRxFifoTable\_0

General

Can RxFifo Identifier 0

📄

255

▼

Can RxFifo Identifier 1

📄

255

▼

Can RxFifo Identifier 2

📄

255

▼

Can RxFifo Identifier 3

📄

255

▼

Can RxFifo Frame Accepted Mode

📄

Standard

▼

✎

▼

Can RxFifo Filter Mask

📄

2047

▼

✎

▼

Figure 4-11. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = D

4.3.1.26.1 CanControllerIDAcceptanceMode

Table 4-57. CanControllerIDAcceptanceMode

Description	This 2-bit field identifies the format of the elements of the Rx FIFO filter table.
Class	Implementation Specific Parameter
Range	A - D
Default	A
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	<pre>CONST(Can_ControllerDescriptorType, CAN_CONST) ControllerDescriptors_PC[CAN_MAXCONTROLLERCOUNT_0] = { ..... /* ===== Controller Options ===== */ /* ID Acceptance Mode A */ CAN_CONTROLLERCONFIG_IDAM_A_U32   }</pre>
AUTOSAR 4.3 Requirement	NA

4.3.1.26.2 CanIDValue0

Table 4-58. CanIDValue0

Description	<p>Specifies an ID to be used as acceptance criteria for the ID Table0.</p> <p>Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below:</p>
-------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table continues on the next page...

**Table 4-58. CanIDValue0 (continued)**

	Format A - One full ID (standard or extended) per filter element Format B - One full standard ID if the CanTableIDType is Standard or one 14 most significant bit value of Extended ID if the CanTableIDType is Extended Format C - One 8 most significant bit value of Standard or Extended ID.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 2047 for Format A and B— Standard CanTableIDType type 0 – 536870911 for Format A – Extended CanTableIDType type 0 – 1683 for Format B – Extended CanTableIDType type 0 – 255 for Format C
<b>Default</b>	0xF0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	Format C <pre>CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PCCConfig[CAN_MAXTABLEID_0] = { {0x11121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */ 0xffffffff }}  Format B CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= { { {0x1ff807f8, /* CanRxFifoTable_0 of type Standard and formatB for FlexCAN_A */ 0xfff8fff8 }}</pre> Format A <pre>CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= { {0x4001ffe, /* CanRxFifoTable_3 of type Extended and formatA for FlexCAN_A */ 0xffffffe }}</pre>
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies an ID to be used as acceptance criteria for the ID Table 0.

**4.3.1.26.3 CanIDValue1****Table 4-59. CanIDValue1**

<b>Description</b>	Specifies an ID to be used as acceptance criteria for the ID Table1. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below: Format B - One full standard ID if the CanTableIDType is Standard or one 14 most significant bit value of Extended ID if the CanTableIDType is Extended
--------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*Table continues on the next page...*

**Table 4-59. CanIDValue1 (continued)**

	Format C - One 8 most significant bit value of Standard or Extended ID.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 2047 for Format B– Standard CanTableIDType type 0 – 1683 for Format B – Extended CanTableIDType type 0 – 255 for Format C
<b>Default</b>	0xF0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	Format C CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PCCfg[CAN_MAXTABLEID_0] = { {0x11121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */ 0xffffffff }}  Format B CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PBCfg [CAN_MAXTABLEID]= { { {0x1ff807f8, /* CanRxFifoTable_0 of type Standard and formatB for FlexCAN_A */ 0xff8fff8 }} }
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies an ID to be used as acceptance criteria for the ID Table 1.

**4.3.1.26.4 CanIDValue2****Table 4-60. CanIDValue2**

<b>Description</b>	Specifies an ID to be used as acceptance criteria for the ID Table2.  Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below  C- One 8 most significant bit value of Standard or Extended ID.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0xF0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	Format C CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PBCfg [CAN_MAXTABLEID]= { {0x11121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */ 0xffffffff }} }
<b>Autosar 4.3 Requirement</b>	NA



**Note**

Specifies an ID to be used as acceptance criteria for the ID Table 2.

**4.3.1.26.5 CanIDValue3****Table 4-61. CanIDValue3**

<b>Description</b>	Specifies an ID to be used as acceptance criteria for the ID Table3. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below C- One 8 most significant bit value of Standard or Extended ID.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0xF0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	Format C CONST(Can_RxFifoTableIdConfigType, CAN_CONST) Rx FIFO Table ID_PBCfg[CAN_MAXTABLEID]= { {0x11121314, /* CanRx FIFO Table_0 of formatC for FlexCAN_A */ 0xffffffff } }
<b>Autosar 4.3 Requirement</b>	NA

**Note**

Specifies an ID to be used as acceptance criteria for the ID Table 3.

**4.3.1.26.6 CanMBFilterMaskValue****Table 4-62. CanMBFilterMaskValue**

<b>Description</b>	Specifies filter mask value to be used as acceptance criteria for the ID Table. Note: Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below: A- Filtermask value for One full standard or Extended ID based on the CanTableIDType selected. B- Filtermask value for CanIDValue0(bit field : 0-10 for standard ID type and 0-13 for extended ID type) and CanIDValue1 (bit field : 11-21 for standard ID type and 14-27 for extended ID type) C- Filtermask value for CanIDValue0(bit field : 0-7), CanIDValue1(bit field : 8-15), CanIDValue2(bit field : 16-23) and CanIDValue3(bit field : 24-31).
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 – 2047 for Format A Standard CanTableIDType 0 - 536870911 Format A Extended CanTableIDType 0 - 268435455 for Format B Extended CanTableIDType

*Table continues on the next page...*

**Table 4-62. CanMBFilterMaskValue (continued)**

	0 - 4194303 for Format B Standard CanTableIDType 0 – 4294967295 for Format C
<b>Default</b>	2047
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	CONST(Can_RxFifoTableIdConfigType, CAN_CONST) RxFifoTableID_PBCfg[CAN_MAXTABLEID]= {  {0x11121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */, <b>0xFFFFFFFF</b> } }
<b>Autosar 4.3 Requirement</b>	NA

#### 4.3.1.26.7 CanTableIDType

**Table 4-63. CanTableIDType**

<b>Description</b>	Specifies whether extended or standard frames are accepted into the FIFO.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	Standard, Extended
<b>Default</b>	Standard
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	NA

### 4.3.2 Can HardwareObject

The screenshot shows the 'CanTTHardwareObjectTrigger' configuration window. It has a 'General' tab selected. The parameters are as follows:

- FD padding value (0 -> 255): 0
- Can Implementation Type: BASIC
- Can ID Message Type: STANDARD
- CanIdValue (Message ID) (0 -> 4294967295): 0
- Can ID Bits Local Priority (0 -> 7): 0
- Can Object ID (MB Handle) (0 -> 65535): 0
- Can MB Type: RECEIVE
- Can Controller Reference: /Can/Can/CanConfigSet\_0/CanController\_0
- Can Filter Mask Reference: /Can/Can/CanConfigSet\_0/CanController\_0/CanFilterMask\_0
- Can MainFunction RW Period Reference: /Can/Can/CanGeneral/CanMainFunctionRWPeriods\_0

**Figure 4-12. Can HardwareObject**

#### Note

When CanControllerRxFifoEnable for a controller is set to true then maximum number of hardware objects to be configured for that controller is 26.

When the FEN bit is set in the MCR register, the memory area from 0x80 to 0xDC (which is normally occupied by MBs 0 to 5) is used by the reception FIFO engine.

For reading data received from Fifo it should be used as the reading from MB0.

The MBs configuration at Can\_Init() level will start to configure classic MBs from the MB index 6, because the space of MBs 0 to 5 is reserved for RxFifo.

The maximum of HOH might be different in the following cases:

- When FIFO is enable, a specified number of MBs will be used by FIFO. Therefore, the total number of HOH should be smaller than the total MBs available. For example, in S32K14X, Flexcan A has 32 MBs. If you using this controller with FIFO, they can not configure up to 32 HOH. The real number depend on the filter size.

- When FD is enable, and the payload is different than 8 bytes, the number of MBs usable is smaller than available. For example, in S32K14X, FlexCan A has 32 MBs. However, if using FD with 64 bytes of data, you can only configure 7 HOHs.

- In an other special case, when the number of MBs are different between controllers, the current driver only check with the maximum one. Therefore, it can not prevent you configure over the number for the controller with smaller number of MBs. For example, in S32K14X, FlexCan A has 32 MBs, FlexCan B and C has 16 MBs. You could configure more than 16 MBs for FlexCan B and C although Flexcan B and C have only 16 ones, but you shouldn't configure more than 16 MBs for FlexCan B and C. Please check the maximum MBs amount for each controller in Reference Manual.

### 4.3.2.1 CanFdPaddingValue

**Table 4-64. CanFdPaddingValue**

<b>Description</b>	This value it is the padding value when FD it is used.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 255
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBcfg.c
<b>Source Representation</b>	<pre>static CONST(Can_MBCfgObjectType, CAN_CONST) MessageBufferConfigs0_PB[CAN_MAXMBCOUNT_0] = { ..... #if (CAN_FD_MODE_ENABLE == STD_ON){</pre>

*Table continues on the next page...*

**Table 4-64. CanFdPaddingValue (continued)**

	<pre> ..... (uint8)0x0, /**&lt; @brief Padding value for MB &gt; 8 bytes */ ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00485

**Note**

Holds the handle ID of HRH or HTH.

**4.3.2.2 CanHandleType****Table 4-65. CanHandleType**

<b>Description</b>	Specifies the type (Full-Can or Basic-Can) of the hardware object.
<b>Class</b>	Autosar Parameter
<b>Range</b>	BASIC, FULL
<b>Default</b>	BASIC
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_MBConfigObjectType, CAN_CONST) MessageBufferConfigs_PC[CAN_MAXMBCOUNT_0] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uldMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U, </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00323

### 4.3.2.3 CanIdType

**Table 4-66. CanIdType**

<b>Description</b>	Specifies whether the IdValue is of type: standard identifier, extended identifier, mixed mode.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Standard, Mixed, Extended
<b>Default</b>	Standard
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_MBCfgObjectType, CAN_CONST) MessageBufferConfigs_PC[] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uldMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U, </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00065

#### Note

Specifies whether the IdValue is of type: - standard identifier (ID - 11 bits length), - extended identifier (ID - 29 bits length), - mixed mode (standard or extended).

### 4.3.2.4 CanIdValue

**Table 4-67. CanIdValue**

<b>Description</b>	Specifies (together with the filter mask) the identifiers that pass the hardware filter for of RX objects.
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 .. 4294967295
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONSTCONST(Can_MBCfgObjectType, CAN_CONST) MessageBufferConfigs_PC[] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * </pre>

*Table continues on the next page...*

**Table 4-67. CanIdValue (continued)**

	<pre> /* uldMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U, </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00325

**Note**

Specifies (together with the filter mask)- the identifiers range that passes the hardware filter for of RX objects. Parameter ranges from 0 to 0x7FF (11 bits) for Standard IDs and 0 to 0x1FFFFFFF (29 bits) for Extended IDs. User can assign any code to this parameter, but must to respect the above rule related to Standard/Extended IDs.

**4.3.2.5 CanMBPrio****Table 4-68. CanMBPrio**

<b>Description</b>	This field is used when MCR[LPRIO_EN] is set and makes sense only for Tx MBs.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	0 - 7
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_MBConfigObjectType, CAN_CONST) MessageBufferConfigs_PC[] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uldMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, </pre>

*Table continues on the next page...*

**Table 4-68. CanMBPrio (continued)**

	/* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
<b>Autosar 4.3 Requirement</b>	NA

**Note**

MBCS[PRIO]: Local priority. This 3-bit field is used when MCR[LPRIO\_EN] is set and makes sense only for TX buffers. If CTRL[LBUF] is set this field is not used. These bits are not transmitted. They are appended to the regular ID to define the transmission priority.

**4.3.2.6 CanObjectId****Table 4-69. CanObjectId**

<b>Description</b>	Holds the handle ID of Hrh or Hth. The value of this parameter is unique in a given CAN Driver and should start with 0 and continue without any gaps. The Hrh and Hth IDs are defined under two different name-spaces. Examples: Hrh0-0, Hrh1-1, Hth0-2, Hth1-3
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 .. 65535
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CanA0_RX0 0U /* RECEIVE object */
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00326

**Note**

Holds the handle ID of HRH or HTH.

**4.3.2.7 CanObjectType****Table 4-70. CanObjectType**

<b>Description</b>	Specifies if the HardwareObject is used as Transmit or as Receive object
<b>Class</b>	Autosar Parameter
<b>Range</b>	Transmit, Receive
<b>Default</b>	Receive

*Table continues on the next page...*

**Table 4-70. CanObjectType (continued)**

<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_MBCfgObjectType, CAN_CONST) MessageBufferConfigs_PC[] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uldMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U, </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00327

**Note**

Specifies if the HardwareObject is used as Transmit or as Receive object.

**4.3.2.8 CanControllerRef****Table 4-71. CanControllerRef**

<b>Description</b>	This associates the hardware object to the CAN controller that uses this hardware object.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Integer
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_MBCfgObjectType, CAN_CONST) MessageBufferConfigs_PC[] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uldMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ </pre>

*Table continues on the next page...*



**Table 4-71. CanControllerRef (continued)**

	RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00322

**Note**

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

**4.3.2.9 CanFilterMaskRef****Table 4-72. CanFilterMaskRef**

<b>Description</b>	Describes a mask for hardware-based filtering of CAN identifiers. It shall be distinguished between: - Standard identifier mask, - Extended identifier mask. The CanFilterMaskRef is omitted for 1) CanHardwareObjects with CanObjectType set to TRANSMIT 2) CanHardwareObjects with CanObjectType set to RECEIVE if only a single Can ID shall be received via this CanHardwareObjects (i.e. exact match with CanIdValue requested when CanHandleType is configured as FULL)
<b>Class</b>	Autosar Parameter
<b>Range</b>	Integer
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_MBCfgObjectType, CAN_CONST) MessageBufferConfigs_PC[] = { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* uIdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U, </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00321

**Note**

Reference to the filter mask that is used for hardware filtering together with the CAN\_ID\_VALUE. This value is used as acceptance masks for ID filtering in RX MBs and the FIFO. This Reference is not used for MCAN implementaion. Since the message buffers cannot configured for reception, the values entered in the filter reference table will be ignored.

**4.3.2.10 CanMainFunctionRWPeriodRef**

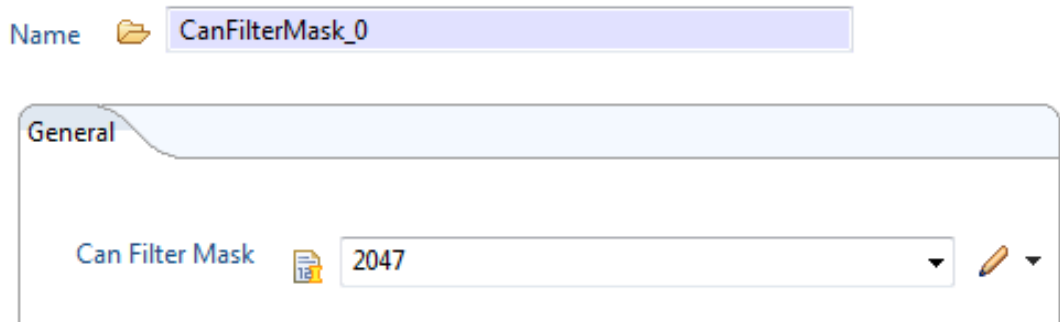
**Table 4-73. CanMainFunctionRWPeriodRef**

Description	Reference to CAN Controller to which the HOH is associated to.
Class	Autosar Parameter
Range	NA
Default	NA
Container Name	CanMainFunctionRWPeriods{CAN_MAIN_FUNCTION_RWPERIODS}
Autosar 4.3 Requirement	ECUC_Can_00437

**Note**

This parameter describes the period for cyclic call to Can\_MainFunction\_Write and Can\_MainFunction\_Read . Unit is seconds. Different poll-cycles will be configurable if more than one CanMainFunctionWritePeriod or Can\_MainFunction\_ReadPeriod are configured. In this case multiple Can\_MainFunction\_Write() or Can\_MainFunction\_Read() will be provided by the CAN Driver module.

**4.3.3 CanFilterMask**



**Figure 4-13. CanFilterMask**

### 4.3.3.1 CanHwFilterMask

**Table 4-74. CanHwFilterMask**

<b>Description</b>	Reference to the filter mask that is used for hardware filtering together with the CAN_ID_VALUE
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 4294967296
<b>Default</b>	2047
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_IdType, CAN_CONST) Can_FilterMasks0_PB[CAN_MAXFILTERCOUNT_0] = { ..... /* FilterMasks_PC[0], "CanFilterMask_0" */ 0x7ffU, ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00066

#### Note

Describes a mask for hardware-based filtering of CAN identifiers.

### 4.3.3.2 CanHwFilterCode

**Table 4-75. CanHwFilterCode**

<b>Description</b>	Reference to the filter mask that is used for hardware filtering together with the CAN_ID_VALUE
<b>Class</b>	Autosar Parameter
<b>Range</b>	0 - 4294967296
<b>Default</b>	2047
<b>Source File</b>	Can_Cfg.c, Can_PBCfg.c
<b>Source Representation</b>	<pre> CONST(Can_IdType, CAN_CONST) Can_FilterMasks0_PB[CAN_MAXFILTERCOUNT_0] = { ..... /* FilterMasks_PC[0], "CanFilterMask_0" */ 0x7ffU, ..... } </pre>
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00066

Note

Describes a mask for hardware-based filtering of CAN identifiers.

4.3.4 Can-General Parameters

CanGeneral parameters, their possible values and meaning are described in the following text. CanGeneral parameters are implemented as preprocessor defines.

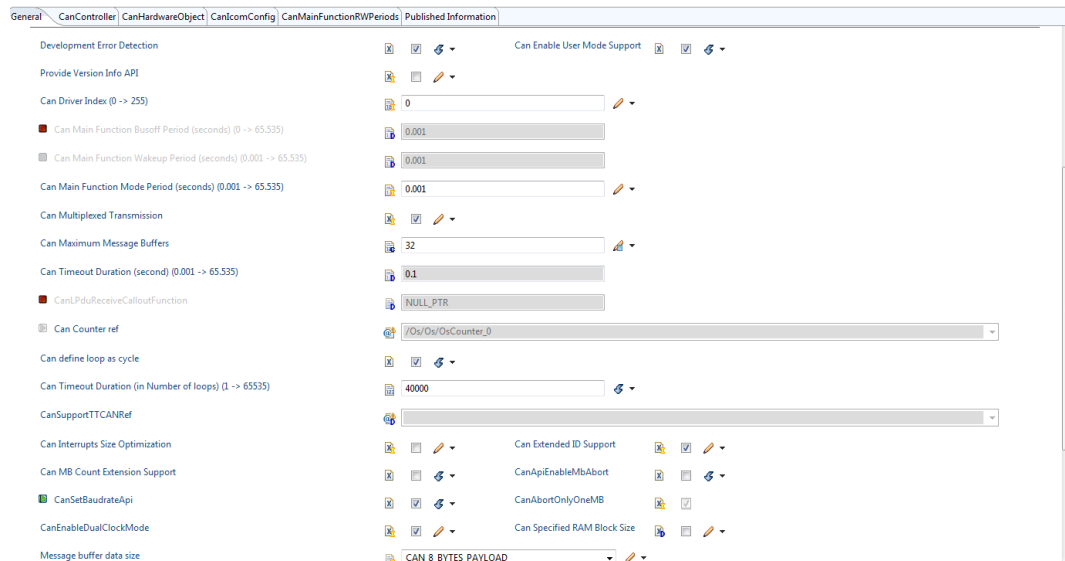


Figure 4-14. Can General Parameters

4.3.4.1 CanDevErrorDetect

Table 4-76. CanDevErrorDetect

Description	Switches the Development Error Detection and Notification ON or OFF.
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_DEV_ERROR_DETECT STD_OFF / STD_ON
Autosar 4.3 Requirement	ECUC_Can_00064

Note

Setting this control to false will generate code that is reduced, but some Autosar requirements are not tested (no Det errors are reported in this way).

### 4.3.4.2 CanVersionInfoApi

**Table 4-77. CanVersionInfoApi**

<b>Description</b>	Defines whether version information reporting should be included at compile time (STD_ON) or excluded (STD_OFF).
<b>Class</b>	Autosar Parameter
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_VERSION_INFO_API STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00106

#### Note

Setting this define to STD\_OFF will decrease the code size, but not support for Can\_GetVersionInfo API will be available.

### 4.3.4.3 CanIndex

**Table 4-78. CanIndex**

<b>Description</b>	Specifies the Instance ID of this module instance. If only one instance is present it shall have the ID 0.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Integer
<b>Default</b>	0
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_INSTANCE 0
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00320
<b>NOTE</b>	<i>This parameter is transmitted as the second parameter to the Det_Reporterror function.</i>

#### Note

This parameter is transmitted as the second parameter to the Det\_Reporterror function.

### 4.3.4.4 CanMainFunctionBusOffPeriod

**Table 4-79. CanMainFunctionBusOffPeriod**

<b>Description</b>	Describes the period for cyclic call to Can_MainFunction_Busoff (in seconds).
<b>Class</b>	Autosar Parameter

*Table continues on the next page...*

**Table 4-79. CanMainFunctionBusOffPeriod  
(continued)**

<b>Range</b>	0.001 .. 65.535
<b>Default</b>	0.001
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_MAINFUNCTION_PERIOD_BUSOFF 0.001U
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00355

**Note**

This parameter is optional. The period value is not used in the Can driver. It should be exported to SchM for using it when polling mode is selected.

**4.3.4.5 CanMainFunctionModePeriod****Table 4-80. CanMainFunctionModePeriod**

<b>Description</b>	Describes the period for cyclic call to Can_MainFunction_Mode (in seconds).
<b>Class</b>	Autosar Parameter
<b>Range</b>	0.001 .. 65.535
<b>Default</b>	0.001
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_MAINFUNCTION_MODE_PERIOD 0.001U
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00376

**Note**

This period value is not used in the Can driver. It should be exported to SchM for using it when polling mode is selected.

**4.3.4.6 CanMultiplexedTransmission****Table 4-81. CanMultiplexedTransmission**

<b>Description</b>	Defines whether support for multiplex transmission should be included at compile time (STD_ON) or excluded (STD_OFF).
<b>Class</b>	Autosar Parameter
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_MULTIPLEXED_TRANSMISSION STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00095

**Note**

When this define is set to STD\_ON the multiplex transmission is used. Multiplex transmission means to send a Can message from any MB that is free, MB that has CanObjectId equal to the one transmitted as parameter to Can\_Write. Multiple MBs can have the same ObjectID.

**4.3.4.7 CanTimeoutDuration****Table 4-82. CanTimeoutDuration**

<b>Description</b>	Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops.
<b>Class</b>	Autosar Parameter
<b>Range</b>	Integer
<b>Default</b>	2000
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_TIMEOUT_DURATION 20U
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00113

**Note**

This value represents a number of finite "while-loops" that the Driver can wait until a hardware set is configured. There is no correspondence between this number of loops and the number of uC cycles.

**Note**

Recommendation about the minimum timeout duration to enter Freeze mode. According to procedure to enter Freeze mode, it need to poll until Freeze Mode Acknowledge is set to 1 or the time out is reached. The minimum timeout duration be equivalent to: a. 730 CAN Nominal bits if CAN FD Operation is enabled (CAN bits calculated at arbitration bit rate), b. 180 CAN bits if CAN FD Operation is disabled.

**4.3.4.8 CanLPduReceiveCalloutFunction****Table 4-83. CanLPduReceiveCalloutFunction**

<b>Description</b>	Specifies the name of the callout function.
<b>Class</b>	Autosar Parameter

*Table continues on the next page...*

**Table 4-83. CanLPduReceiveCalloutFunction  
(continued)**

<b>Range</b>	NA
<b>Default</b>	NULL_PTR
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	extern FUNC(boolean, COM_APPL_CODE) 'LPduCalloutFunction_name'(uint8 Hrh, Can_IdType CanId, uint8 CanDlc, const uint8 *CanSduPtr);
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00434

**Note**

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.

**Note**

In order to use LPDU callout Can\_GeneralTypes.h need to be included in the file where the callout function is defined. The following files need to be included prior to include Can\_GeneralTypes.h - ComStack\_Cfg.h and Can\_Cfg.h

**4.3.4.9 CanCodeSizeOptimization****Table 4-84. CanCodeSizeOptimization**

<b>Description</b>	Enables optimization of interrupt service routines for code size.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_ISROPTCODESIZE STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	NA

**Note**

The value of this parameter doesn't matter for this platform because there is a single interrupt handler for each CAN controller.



### 4.3.4.10 CanExtendedIdSupport

**Table 4-85. CanExtendedIdSupport**

<b>Description</b>	Enables support of Extended/Mixed mode.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	True
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_EXTENDEDID STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	NA

#### Note

This parameter permit to enable the Message ID representation on uint32 size variable, else uint16 is used.

### 4.3.4.11 CanMBCountExtensionSupport

**Table 4-86. CanMBCountExtensionSupport**

<b>Description</b>	Enables support of more than 255 Can Hardware Objects.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_MBCOUNTTEXTENSION STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	NA

#### Note

This parameter permit to declare more than 255 MBs in the configuration. If total MBs for all controllers exceed 255 then this parameter must be on.

### 4.3.4.12 CanApiEnableMbAbort

**Table 4-87. CanApiEnableMbAbort**

<b>Description</b>	Enables an additional API.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.h

*Table continues on the next page...*

**Table 4-87. CanApiEnableMbAbort (continued)**

<b>Source Representation</b>	#define CAN_API_ENABLE_ABORT_MB STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter value is considered only if "CanHardwareCancellation" is true.

**4.3.4.13 CanEnableDualClockMode****Table 4-88. CanEnableDualClockMode**

<b>Description</b>	Enables support for dual clock API. Can controller is able to run on the same baudrate over CAN bus using 2 different source clocks that can be changed.
<b>Class</b>	Implementation Specific Parameter
<b>Range</b>	True, False
<b>Default</b>	False
<b>Source File</b>	Can_Cfg.h
<b>Source Representation</b>	#define CAN_DUAL_CLOCK_MODE STD_OFF / STD_ON
<b>Autosar 4.3 Requirement</b>	NA

**Note**

This parameter is visible and configurable only if Can.CanConfig.DualClockMode=STD\_ON from Resource files.

**4.3.4.14 CanCounterRef****Table 4-89. CanCounterRef**

<b>Description</b>	Contains a reference to the counter.
<b>Class</b>	Autosar Parameter
<b>Range</b>	NA
<b>Default</b>	Reference to [ OsCounter ]
<b>Source File</b>	NA
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00431

**Note**

This parameter contains a reference to the counter, which is used by the CAN driver.

### 4.3.4.15 CanSupportTTCANRef

**Table 4-90. CanSupportTTCANRef**

<b>Description</b>	Refers to CanIfSupportTTCAN parameter in the CAN Interface Module configuration.
<b>Class</b>	Autosar Parameter
<b>Range</b>	NA
<b>Default</b>	NA
<b>Source File</b>	NA
<b>Source Representation</b>	NA
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00430

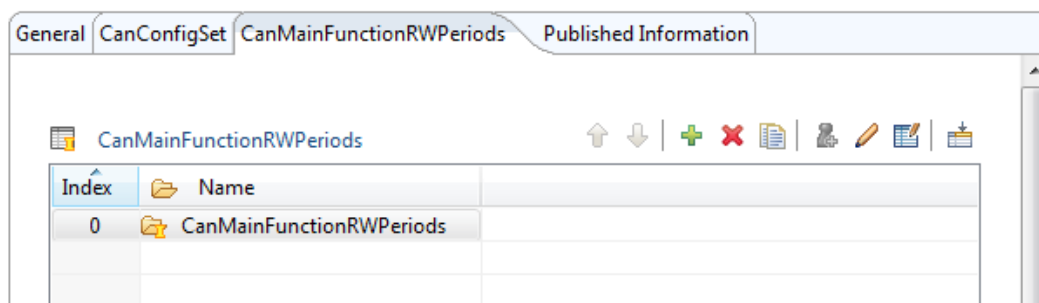
#### Note

The CanIfSupportTTCAN parameter defines whether TTCAN is supported. This requirement is not supported by Can Driver.

### 4.3.4.16 CanMainFunctionRWPeriods

**Table 4-91. CanMainFunctionRWPeriods**

<b>Container Name</b>	CanMainFunctionRWPeriods{CAN_MAIN_FUNCTION_RWPERIODS}
<b>Description</b>	Reference to CAN Controller to which the HOH is associated to.
<b>Class</b>	Autosar Parameter
<b>Autosar 4.3 Requirement</b>	ECUC_Can_00437



**Figure 4-15. CanMainFunctionRWPeriods**

#### Note

This parameter describes the period for cyclic call to Can\_MainFunction\_Write and Can\_MainFunction\_Read . Unit is seconds. Different poll-cycles will be configurable if more than one CanMainFunctionWritePeriod or Can\_MainFunction\_ReadPeriod are configured. In this case

multiple Can\_MainFunction\_Write() or Can\_MainFunction\_Read() will be provided by the CAN Driver module.

4.3.4.16.1 CanMainFunctionPeriod

Table 4-92. CanMainFunctionReadPeriod

Description	This parameter describes the period for cyclic call to Can_MainFunction_Read/Write. Unit is seconds.
Class	Autosar Parameter
Range	0.001 .. 65.535
Default	NA
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_READ/WRITE 0.001U
Autosar 4.3 Requirement	ECUC_Can_00356

Note

Different poll-cycles will be configurable if more than one CanMainFunctionReadPeriod is configured. In this case multiple Can\_MainFunction\_Read/Write() will be provided by the CAN Driver module.

4.4 Form CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.

The screenshot shows the Treso Plugin configuration interface. At the top, there's a 'Can' object selected. Below it, a tabbed interface shows the 'Published Information' tab. Under this tab, the 'CommonPublishedInformation' form is displayed. It contains several fields, each with a lightbulb icon indicating a default or recommended value:

- Ar Release Major Version: 4
- Ar Release Minor Version: 3
- Ar Release Revision Version: 1
- ModuleId: 80
- SwMajorVersion: 1
- SwMinorVersion: 0
- SwPatchVersion: 1
- VendorApilnfix: (empty)
- VendorId: 43

Figure 4-16. Treso Plugin snapshot for CommonPublishedInformation form.

#### 4.4.1 ArReleaseMajorVersion (CommonPublishedInformation)

Major version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-93. Attribute ArReleaseMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	4
Invalid	Range <div>&gt;=4</div> <div>&lt;=4</div>

#### 4.4.2 ArReleaseMinorVersion (CommonPublishedInformation)

Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

**Table 4-94. Attribute ArReleaseMinorVersion (CommonPublishedInformation) detailed description**

Property	Value
Label	AUTOSAR Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	3
Invalid	Range >=3 <=3

### 4.4.3 ArReleaseRevisionVersion (CommonPublishedInformation)

Revision version number of AUTOSAR specification on which the appropriate implementation is based on.

**Table 4-95. Attribute ArReleaseRevisionVersion (CommonPublishedInformation) detailed description**

Property	Value
Label	AUTOSAR Release Revision Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

### 4.4.4 ModuleId (CommonPublishedInformation)

Module ID of this module from Module List.

**Table 4-96. Attribute ModuleId (CommonPublishedInformation) detailed description**

Property	Value
Label	Module Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false

*Table continues on the next page...*

**Table 4-96. Attribute ModuleId (CommonPublishedInformation) detailed description (continued)**

Property	Value
Default	80
Invalid	Range >=80 <=80

#### 4.4.5 SwMajorVersion (CommonPublishedInformation)

Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

**Table 4-97. Attribute SwMajorVersion (CommonPublishedInformation) detailed description**

Property	Value
Label	Software Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

#### 4.4.6 SwMinorVersion (CommonPublishedInformation)

Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

**Table 4-98. Attribute SwMinorVersion (CommonPublishedInformation) detailed description**

Property	Value
Label	Software Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range >=0 <=0

#### 4.4.7 SwPatchVersion (CommonPublishedInformation)

Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

**Table 4-99. Attribute SwPatchVersion (CommonPublishedInformation) detailed description**

Property	Value
Label	Software Patch Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

#### 4.4.8 VendorApiInfix (CommonPublishedInformation)

In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows:

<ModuleName>\_<VendorId>\_<VendorApiInfix><Api name from SWS>. E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can\_Write defined in the SWS will translate to Can\_123\_v11r456Write. This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

**Table 4-100. Attribute VendorApiInfix (CommonPublishedInformation) detailed description**

Property	Value
Label	Vendor Api Infix
Type	STRING_LABEL
Origin	Custom
Symbolic Name	false
Default	
Enable	false



### 4.4.9 VendorId (CommonPublishedInformation)

Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

**Table 4-101. Attribute VendorId (CommonPublishedInformation) detailed description**

Property	Value
Label	Vendor Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	43
Invalid	Range <div> <div>&gt;=43</div> <div>&lt;=43</div> </div>



**How to Reach Us:****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2019 NXP B.V.

Document Number UM2CANASR4.3 Rev0001R1.0.1  
Revision 1.0